State University of New York at Albany

COLLEGE OF ARTS AND SCIENCES

The Dissertation submitted by

Tushar Saxena

under the title

EFFICIENT VARIABLE ELIMINATION USING RESULTANTS

has been read by the undersigned. It is hereby recommended for acceptance to the Faculty of the University in partial fulfillment of the requirement for the degree of Doctor of Philosophy.

(Deepak Kapur)	(Date)
(Harry B. Hunt III)	(Date)
(Paliath Narendran)	(Date)
(Lakshman Y.N.)	(Date)

Recommended by the Department of Computer Science.

____________________, Chair.

(Signed)

Recommendation accepted by the Dean of Graduate Studies for the Graduate Academic Council.

(Signed)

(Dated)

EFFICIENT VARIABLE ELIMINATION USING RESULTANTS

by

Tushar Saxena

EFFICIENT VARIABLE ELIMINATION USING RESULTANTS

by

Tushar Saxena

A Dissertation
Submitted to the State University of New York at Albany
in Partial Fulfillment of
the Requirements for the Degree of
Doctor of Philosophy

College of Arts and Sciences
Department of Computer Science
1997

Abstract

A new method for eliminating variables from polynomial equations is proposed, analyzed, evaluated and applied. Problems from many applications involve computation of resultants – polynomials obtained after eliminating variables from polynomials. Therefore effective elimination procedures require efficient methods to compute resultants, especially those exploiting sparsity which typically exists in most *real-world* applications.

A new method for computing resultants is proposed. This method exploits the sparsity in the problem, despite being based on a classical formulation by Dixon. It also extends the Dixon formulation for bi-degree polynomials to compute resultants of most multivariate polynomial systems, while *implicitly* exploiting the sparse structure of the input polynomial system. Our work, *for the first time*, links the classical Dixon formulation to the modern line of sparsity analysis based on Newton polytopes. This link enables us to (i) devise an algorithm for directly interpolating the resultant, (ii) tightly bound the complexity of our algorithm by expressing it in terms of the volumes of the input polytopes and (iii) prove that for most classes of polynomial systems, the proposed method is more efficient than other prevalent resultant algorithms.

Practicality of the method is demonstrated by applying its efficient implementation to problems from a variety of application domains such as Computer Vision, Robotics, Solid Modeling and Geometric Reasoning. Ours was the first solution to some of these problems, and for others, the fastest yet.

These results demonstrate, both theoretically and empirically, that our method is more effective than other existing variable elimination techniques.

Acknowledgements

I am most indebted to my advisor, Deepak Kapur, for being a rich source of ideas, insightful advise and inspiration throughout my research. Without his direction, guidance and encouragement this thesis would not have been possible. I consider myself fortunate for having an opportunity to work with him.

The research in this thesis also builds upon the initial collaborative work we did with Yang Lu. His early influence on this research was immeasurably significant. I thank him for providing me that fruitful opportunity. I am also grateful to Lakshman Y.N. for his interest in my research. I am indebted to him for taking time out of his busy schedule so many times to listen to my ideas and improve them by challenging me to be rigorous and precise. I learnt several mathematical concepts as a direct result of my interactions with him.

My heartfelt thanks to Harry Hunt and Paliath Narendran for serving on my thesis committee and for providing many insightful and extremely useful comments.

I am also grateful to the United States Air Force Office of Scientific Research and the National Science Foundation for partially supporting this research under grants AFOSR-91-0361 and CCR-9622860 respectively.

My sincere appreciation goes out to Sandeep Shukla who made last few years particularly enjoyable.

Finally, I am humbled by the unceasing support I received from my parents and brother for anything I wanted to pursue. I affectionately dedicate this thesis to my father and mother. It is their love and sacrifices which made this possible in the first place.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation for Elimination Theory

To give a flavor of elimination theory and motivate its study, we start by discussing the following problem which involves geometric reasoning. Heymann in [25] posed the question of whether a triangle can be drawn using only a compass and a ruler, given the lengths of any three of its angle bisectors. One specific case of this problem can be mathematically formulated as follows.

Let ABC (see Figure 1.1) be the triangle in question, in which a, b and c are the lengths of the three sides, a_i the internal bisector of angle A, and a_e and b_e be the external bisectors of angles A and B, respectively. Heymann's question translates to whether ABC can be drawn, given a_i, a_e and b_e.

Equations (1.1) express the bisectors in terms of the sides and can be easily derived using high-school mathematics. To construct the triangle, one must be able to construct line segments of length a, b and c. From Corollary[1] 2 of Theorem 5.4.1 in [24], it follows that if the relationship between the three bisectors and a is an irre-

[1]The corollary says that if a real number α satisfies an irreducible polynomial over the field of rational of degree k, and if k is not a power of two, then a line segment of length α cannot be constructed using a compass and a ruler.

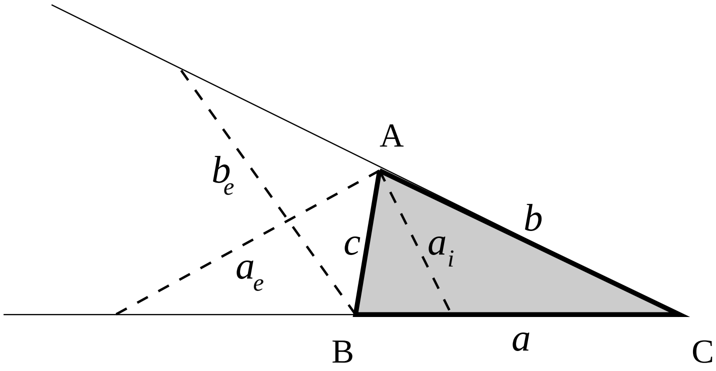

Figure 1.1: Triangle for Heymann's Problem

$$\left.\begin{aligned} a_i^2 &= \frac{cb(c+b-a)(c+b+a)}{(b+c)^2} \\ a_e^2 &= \frac{cb(a+b-c)(c-b+a)}{(c-b)^2} \\ b_e^2 &= \frac{ac(a+b-c)(c+b-a)}{(c-a)^2} \end{aligned}\right\} (1.1)$$

$$\left.\begin{aligned} a_i^2(b+c)^2 - cb(c+b-a)(c+b+a) &= 0 \\ a_e^2(c-b)^2 - cb(a+b-c)(c-b+a) &= 0 \\ b_e^2(c-a)^2 - ac(a+b-c)(c+b-a) &= 0 \end{aligned}\right\} (1.2)$$

Figure 1.2: Equations for Heymann's Problem

$$
\left[\begin{matrix}
0 & 0 & 0 & 3600+136a^2 & -128 & -136 \\
0 & 0 & 0 & 0 & 0 & 0 \\
-136a^2-3600 & 0 & 256a^2 & 1664a^2-3600+68a^4 & -64a^2+128 & 136-52a^2 \\
0 & -136a^2-3600 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -7200+628a^2+34a^4 & -32a^2+256 & -34a^2+272 \\
0 & 128a^2 & -256a+32a^3 & -104a^3+1088a-4a^5 & 0 & 4a^3+36a \\
0 & 136a^2 & 628a+68a^3 & -32a^3+256a & -8a^3-36a & -32a \\
0 & 136a^2 & -1172a & -256a+32a^3 & 68a & 0 \\
136 & 0 & 0 & 136-68a^2 & 0 & 0 \\
0 & 0 & 3600-1664a^2-68a^4 & -256a^2 & 240a^2-544+8a^4 & -128+64a^2 \\
136 & 0 & 0 & 136-84a^2 & 0 & 0 \\
128 & 0 & 16a^2 & -64a^2+128 & -8a^2+16 & 0 \\
0 & 0 & 0 & 4a^3-32a & 0 & -4a
\end{matrix}\right.
$$

$$
\left.\begin{matrix}
0 & 0 & 0 & 0 & -136 & 0 & 0 \\
0 & 0 & 0 & -34a^2-900 & 0 & 34 & 32 \\
-628a^2+7200-34a^4 & 34a^3-272a & -256a+32a^3 & -1172a & 136-68a^2 & 34a & 32a \\
0 & 128 & 16a^2+544 & -128 & 0 & 0 & -16 \\
3600+136a^2 & -136a & -128a & -136a & -34a^2+272 & 0 & 0 \\
-544a-16a^3 & 136-52a^2 & 0 & 188a^2-408+8a^4 & -68a-4a^3 & 0 & 0 \\
-128a & 0 & 136-84a^2 & -128+64a^2 & 0 & 0 & 8a^2-16 \\
128a & 0 & 136-68a^2 & -64a^2+128 & 32a & 0 & 0 \\
34a^2-272 & -34a & 0 & 34a & 0 & 0 & 0 \\
0 & 0 & 1800a+68a^3 & 512a-64a^3 & -128+64a^2 & -32a & -104a-8a^3 \\
34a^2-272 & 34a & -64a & 134a-4a^3 & 8a^2-16 & -4a & 0 \\
-256+32a^2 & 0 & -168a & 64a & 0 & 0 & 8a \\
16a & 0 & 0 & -8a^2+16 & 4a & 0 & 0
\end{matrix}\right]
$$

Figure 1.3: Resultant Matrix for Heymann's Problem

ducible polynomial whose degree in a is *not* 2^m for some integral m, then the answer to Heymann's question is in the negative. In other words, we can solve Heymann's problem if we can:

Derive the relationship purely between a, a_i, a_e and b_e.

A straight-forward way of accomplishing this derivation is by eliminating b and c from Equations (1.2) (which follow from Equations (1.1)). Computationally, this is not an easy task and may require quite a bit of computational resources as well as innovative techniques. It is such techniques that are the object of study under the general subject of *Elimination Theory*. Elimination theory studies and develops efficient techniques for symbolically solving systems of non-linear polynomial equations.

To give a flavor of the contributions of this thesis, we give a solution to Heymann's problem using a technique developed in this thesis. It can be shown using the results of this thesis that the polynomial in a, a_i, a_e and b_e which is the result of eliminating b and c from the three equations (1.2) is in fact the determinant of a 13×13 matrix (whose entries are polynomials in a, a_i, a_e and b_e). Since this matrix is very large and we are anyway interested only in knowing the degree of its determinant in a, we give the matrix after substituting $a_i = 3, a_e = 5$ and $b_e = 2$ in Figure 1.3. Its determinant has degree 20 in a. Since 20 is not an integral power of 2, the answer to Heymann's problem is in the negative. Thus, in general, it is impossible to draw a triangle using just a compass and a ruler, if the lengths of any three of its angle bisectors are given.

1.1.1 Elimination Theory

Elimination theory is the study of techniques and algorithms for eliminating variables from a set of polynomial equations. In the previous section we demonstrated how one problem involving geometric reasoning can be solved using elimination theory. In fact, elimination of variables is a fundamental problem which often comes up in many areas of mathematics, engineering, physical and computer sciences. Many practical problems in robotics, computer vision, computational biology, solid modeling, mechanical and chemical engineering, thermodynamics, physics etc. can be reformulated as a problem of eliminating a subset of variables from a set of polynomial equations. For example, Hoffman in [26] posed the challenging problem of deriving the implicit equation (in x, y and z coordinates) of a bicubic surface (which is plotted in Figure 1.4) from its parameterization. The problem of deriving implicit equations from parameterizations is known as the *implicitization problem*, and can be reformulated as an elimination problem. The bicubic implicitization problem posed by Hoffman translates to the problem of eliminating variables s and t from the fol-

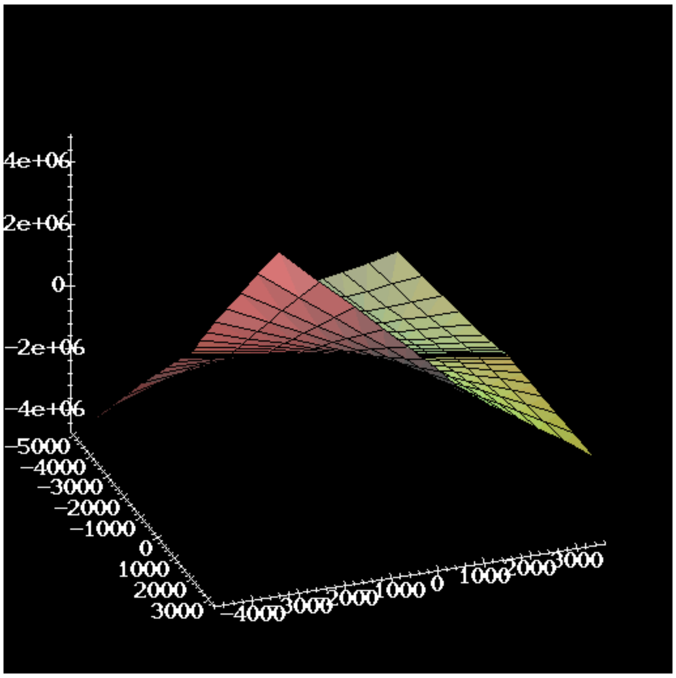

Figure 1.4: A Bicubic Surface

lowing three polynomials:

$$
\begin{aligned}
q_1 &= 3\,t\,(t-1)^2 + (s-1)^3 + 3\,s - x \\
q_2 &= 3\,s\,(s-1)^2 + t^3 + 3\,t - y \\
q_3 &= -3\,s\left(s^2 - 5\,s + 5\right)t^3 - 3\left(s^3 + 6\,s^2 - 9\,s + 1\right)t^2 \\
&\quad +t\left(6\,s^3 + 9\,s^2 - 18\,s + 3\right) - 3\,s\,(s-1) - z.
\end{aligned}
$$

The problem of eliminating variables from polynomial equations is inherently hard in terms of computational resources. Nevertheless, as computer technology and symbolic manipulation techniques have drastically improved over the last few decades, elimination problems in many real applications have come within the grasp of state-of-the-art computational power. For instance, the bicubic implicitization problem stated above had proven to be a computationally challenging problem when Hoffman [26] posed it in his 1990 paper. He reported that even the state of the art elimination methods in 1990 required time of the order of 10^5 seconds to implicitize the surface, which is quite unacceptable for real life applications. However, today, with the development of fast variable elimination techniques such as those in this thesis, this problem can easily be solved within a few ($< 10^1$) seconds on desktop workstations (the bicubic surface represented by the above equations is plotted in Figure 1.4). Already, the fastest solution to problems such as real-time inverse kinematics for a general *6R* serial robot arm manipulator [33] use specialized techniques from elimination theory. Such successes have inspired a renewed interest in developing efficient methods for eliminating variables and solving nonlinear equations. Evidence to this is provided by recent breakthroughs in elimination theory such as the development of fast algorithms for solving polynomial systems with finitely many solutions [18], resurrection and variations of classical constructive techniques for eliminating variables [5, 33], development of elimination methods which exploit the structure of polynomial systems to solve them efficiently [14, 7, 40], and the development of efficient techniques for numerically solving non-linear systems [35, 42].

Three major techniques for eliminating variables symbolically are:

1. Gröbner basis computations proposed by Buchberger [4],

2. Characteristic set computations proposed by Ritt [37] and

3. Resultant computations based on methods developed early this century by researchers such as Macaulay [32], Cayley [8], Bezout, Dixon [12], Hurwitz etc.

A Gröbner basis of a polynomial ideal is a basis with many useful properties and provides answers to most ideal-theoretic questions about the ideal. The first algorithm to compute Gröbner basis of ideals was given by Buchberger in [4] and since then, many efficient variations have appeared [2]. Among other things, Gröbner bases can be used to find solutions to a set of polynomials, compute projections of their variety, eliminate variables and test polynomials for ideal membership. A variation on the Gröbner basis algorithm for 0-dimensional ideals is what was used by Hoffman to solve the bicubic implicitization problem in 10^5 seconds in 1990 [26].

A characteristic set [10, 44] of a set of polynomials is a *triangular* set of polynomials with *almost* the same set of common solutions as the original. The first algorithm to compute characteristic sets was given by Ritt, and recently Wu [44] resurrected it. Characteristic sets are typically computed by eliminating variables sequentially in some predetermined order using successive pseudo-division of polynomials. Characteristic sets can be used, among other things, to compute projections of varieties, eliminate variables and derive conditions under which a polynomial equation follows from another set of polynomial equations. Their effectiveness in proving geometry theorems is well established. In his book ([10]), Chou details many theorem proving procedures using characteristic sets, and lists hundreds of theorems he was able to automatically prove using his implementation of these procedures. Gao and Wang [20] used specialized characteristic set algorithms to solve Heymann's problem in 19 hours on a SUN 4 workstation.

Resultant is a characterization of the projection of the variety of a given set of polynomials into a smaller set of variables. Methods which compute the resultant can thus be used to eliminate a subset of variables from a set of polynomials.

The interesting property of multivariate resultant methods is that they eliminate n variables *together*, instead of sequentially, from $n+1$ polynomials. The fundamental principle by which multivariate resultant methods work is by reducing a non-linear elimination problem to a linear one. That is why the derivation in Heymann's problem was reduced to computing the determinant in Figure 1.3. Such linearization of nonlinear problems enables the applicability of a vast array of linear algebra and linear equation solving techniques that have been developed in the last two centuries, to non-linear variable elimination. The effectiveness of multivariate resultant methods has been demonstrated in a variety of applications [9, 14, 28, 29, 33, 34, 39], and for some applications only the methods involving resultants offer an acceptable real-time solution [33]. For example, the implicit equation of the bicubic surface can be derived within 50 seconds (compared to 10^5 seconds using Gröbner bases), and Heymann's question can be resolved within 300 seconds using resultant based methods on a SPARC station 10 (compared to 19 hours using Characteristic set type methods).

This thesis proposes, analyzes, evaluates and applies a new method for computing resultants based on a classical formulation by Dixon [12]. Before discussing the contributions of this thesis, we outline the different algorithms available for computing the resultant and some desirable qualities in resultant methods. Here we give an informal perspective on multivariate resultant methods so as to create a setting in which the contributions of this thesis and their importance can be understood. We will formally review the various multivariate resultant methods in detail later in this thesis.

1.1.2 Resultants: An Informal Perspective

There are three major formulations which can be used to compute multivariate resultants – (i) The Macaulay formulation, (ii) The Sparse resultant formulation and (iii)

The Dixon formulation. All three formulations eliminate n variables from $n+1$ non-homogeneous polynomials by constructing resultant matrices (which are different for each formulation, being known as the Macaulay, sparse and Dixon resultant matrices respectively) and extracting the resultant from them.

Macaulay Formulation: Early this century, Macaulay developed a method for computing the resultant in [32] which was resurrected in the 1980s by Canny and other researchers [5, 33]. This method expresses the resultant as the ratio of determinants of the Macaulay resultant matrix and one of its submatrices. One problem that often arises in the classical Macaulay formulation, when dealing with non-generic polynomial systems, is that the denominator submatrix, or the whole Macaulay matrix becomes singular. Canny in [5] provided a solution to this problem by systematically *perturbing* the input polynomial system *away* from such singular matrices. Another problem with Macaulay's formulation is that it uses the traditional Bezout bound on the number of common solutions of a polynomial system to construct the Macaulay matrix. This means that the size of the Macaulay matrix depends on the total degree of the input polynomials. Consequently, if two polynomial systems have the same total degree, their Macaulay matrices are of equal size – even though a substantial number of terms may be missing from one system while the other may be completely dense. Macaulay's formulation does not take the sparsity of input polynomials into account. Consequently, resultant matrices constructed by this formulation are too large, and for most real applications, which typically produce sparse polynomial systems, implementations of this formulation do not yield an answer in an acceptable amount of time. For example, the Macaulay matrix for Heymann's problem is 55×55 and its denominator is singular. Using perturbation techniques on this problem does not yield an answer even after running for a day on our implementation as well as on most computer algebra systems.

Sparse Resultant Formulation: Sparse resultant formulation was developed as an effort to modify the Macaulay formulation so that it exploits the sparsity in input polynomials. This method constructs a sparse resultant matrix, whose determinant is a multiple of the resultant. The size of the sparse resultant matrix depends on the BKK bounds [3] which are sharper because they exploit the structure and sparsity of the polynomials as modeled by their Newton polytopes. As a result, the sparse resultant matrix is smaller than the Macaulay matrix and depends on the volume of the polytopes of the input polynomials, not their total degree. For example, the sparse resultant matrix of Heymann's problem is 34×34 since the input polynomials are not completely dense. The sparse resultant method represents the first attempt at explicitly exploiting the sparsity of input polynomials while computing the resultant.

Classical Dixon Formulation: Bezout and later in 1865, Cayley [8], developed an efficient method to compute the resultant of two univariate polynomials. In 1908, Dixon generalized their method to an arbitrary number of variables. Dixon's method constructs a matrix, called the Dixon matrix, whose determinant is a multiple of the resultant. In its classical form, this method works only on a small class of polynomial systems known as *generic bidegree* systems. Dixon matrices of arbitrary polynomial systems can become rectangular, or even if they are square, they can be singular. For example, since the polynomials in Heymann's problem are neither generic nor bidegree, their Dixon matrix is actually 13×14. So, in general, Dixon's original formulation cannot be used for computing the resultant or a non-zero multiple of the resultant. Since this method was very restrictive, it remained relatively unpopular as compared to the Macaulay formulation. No analysis was done about the sparsity considerations in this formulation either. Since Dixon's method is classical whereas sparsity measures involving Newton polytopes relatively modern, one may conclude that Dixon's method, like Macaulay's, ignores sparsity in input systems. Due to all these hurdles and open questions surrounding Dixon's formulation, it never received wide applicability and acceptance.

This thesis removes some of these hurdles and answers some of these open questions about Dixon's method, thus extending its applicability.

1.2 Thesis Contributions and Overview

1.2.1 Contributions

A New Resultant Method based on the Classical Dixon Formulation: We present a method which extends the applicability of the classical Dixon resultant formulation. This work [29] was done in collaboration with Deepak Kapur and Yang Lu. We include it here because it forms an integral prelude to the other results contained in this thesis.

Dixon's original method fails for a large class of polynomial systems because often the resultant matrix it constructs is rectangular, and a direct computation of the resultant as its determinant is not possible. Even when the Dixon matrix is square, it may be singular and the projection operator vanishes identically, giving no information about the zeros of the polynomial system. We describe a method to extract a non-trivial multiple of the resultant from singular or even rectangular Dixon matrices. Since our method uses the degenerate matrix originally constructed by the classical Dixon formulation, it preserves the structure of the input polynomials and retains the efficiency inherent in the original Dixon formulation. We found that most problems on which the original Dixon formulation was inapplicable can be successfully solved using our extended Dixon formulation.

New Results Linking Sparsity and the Extended Dixon Formulation: Next, this thesis presents new results relating the sparsity in non-homogeneous polynomial systems and computation of their resultant using methods based on Dixon's formulation. We prove that despite being classical, Dixon's method of computing resultants *implicitly* exploits the sparse structure of input polynomials. It is proved that the size of the Dixon matrix, and the complexity of computing the resultant using the extended Dixon formulation is not determined by the total degree of the polynomial system, rather by the structure of the Newton polytopes of the polynomial system. As terms are removed from the input polynomials, our method will exploit this fact to construct smaller matrices. We show how the the Newton polytopes of the input polynomials can be used a-priori to completely determine the structure of their Dixon matrix. In this respect, our results show that extended Dixon formulation bears resemblance to the sparse resultant formulation and is unlike the Macaulay formulation. Our work for the first time links the classical Dixon formulation (developed in 1908) to the modern line of sparsity analysis based on Newton polytopes (developed in the 1970s [3]).

New Bounds on the Complexity of Resultant Computation: Based on the relationship of our method to sparsity in the input polynomials, an exact formula for the size of the Dixon matrix of unmixed polynomial systems is derived in terms of their Newton polytopes. This relationship is exploited to tightly bound the size of the Dixon matrices of multi-homogeneous polynomial systems. Using these results, we prove that the Dixon matrix is smaller than both the Macaulay matrix and the sparse resultant matrix. It is proved that for multi-homogeneous systems, the sparse resultant matrix is larger than the Dixon matrix by $\mathcal{O}((r^r/r!)^l)$, where the variable set is partitioned into r partitions, each containing l variables. We then devise an algorithm which constructs Dixon matrices efficiently using dense polynomial interpolation which exploits these tight bounds on the size of the Dixon matrix. We prove that the complexity of computing the resultant of unmixed systems using our algorithm is smaller than the sparse resultant formulation.

A Sparsity-Preserving Alternative to Perturbation: A big advantage of our technique for extracting resultant from degenerate Dixon matrices is that it does not change the structure of the input polynomials. Techniques such as perturbation [5], which may be useful in case of degenerate resultant matrices, add terms to the input polynomials, thus changing the structure of the input. This may increase the degree of the polynomials and destroy any sparsity that may exist in them. Such an effect can have detrimental consequences on resultant methods which exploit precisely this sparsity to efficiently compute the resultant.

We show that our technique can be modified to extract non-trivial multiples of the resultant from degenerate Macaulay and sparse resultant matrices also. Since our technique does not change the structure of the input polynomials, we offer a *sparsity-preserving* alternative to perturbation. This is especially important for the Dixon and sparse resultant formulations, because it maintains the small size of resultant matrices even in the face of degeneracies due to the fact that our technique does not alter the polytopes of the input polynomials.

Experimental Evaluation of the Extended Dixon Formulation: To demonstrate the practical applicability of the proposed method, this thesis presents empirical evidence which supports its theoretical claims. Resultant matrices may have symbolic entries, and their direct expansion results in huge intermediate expressions. We employ polynomial interpolation techniques to avoid these huge expressions by directly computing the resultant from its values at well chosen points. We implemented our method efficiently (in C^{++}) and used it to solve a number of problems from real applications. These empirical results demonstrated the efficiency of our methods. Our detailed experimental comparison [30] of all available techniques for computing the resultant showed that our extension of Dixon formulation takes the least time on most problems. Our implementation has provided the fastest solution yet to problems in a variety of application domains such as computer vision, robotics, solid modeling and geometric reasoning.

1.2.2 Overview

In the next chapter, we review the various popular methods for computing the resultant. This includes the classical formulations by Sylvester, Macaulay, Bezout-Cayley and Dixon as well as the modern method of computing Sparse resultants. Chapter 2 also discusses some merits of the classical Dixon formulation and points out its shortcomings and reasons for its restrictiveness.

Chapter 3 extends Dixon's classical formulation and demonstrates it using some examples. The applicability of our technique, for extracting a non-trivial multiple of the resultant from non-degenerate resultant matrices, to other resultant formulations is also discussed. Chapter 3 concludes by deriving some loose upper bounds on the size of the Dixon resultant matrix constructed using our extension.

Chapter 4 analyzes the sparsity considerations in our method based on the classical Dixon formulation. First the relationship between the structure of the Dixon polynomial, a polynomial whose coefficients form the entries of the Dixon resultant matrix, and the Newton polytopes is described. Based on this relationship, a tight bound on the size of the Dixon resultant matrix is derived. Chapter 4 concludes by comparing the sparsity considerations in various other formulations with those in Dixon's formulation.

Based on the results of chapter 4, chapter 5 determines the total complexity of computing the resultant using our extension of the classical Dixon formulation. An algorithm which exploits the results of chapter 4 to construct Dixon matrices fast is proposed. It is proved that the time complexity of computing the resultant using this algorithm is better than other popular resultant formulations.

Chapter 6 describes the heuristics we developed and used and the details of an efficient implementation of our method. It also discusses the empirical results obtained by applying our implementation on numerous problems from a variety of application domains. Chapter 7 discusses some problems from Computer Vision, Geometric Reasoning and Modeling, Robotics and Kinematics and Algebra which have been solved successfully by applying our method.

Finally, chapter 8 summarizes this thesis and concludes by pointing to some open questions for future research.

Chapter 2

Review of Resultants

In this chapter we will review the basic themes of the subject matter in this thesis. After introducing the notation and basic definitions in the first section, we describe the general approach which is followed by most multivariate resultant methods – classical as well as modern. In other words, in the second section, we abstract out from specific resultant methods and present their essence. Sections three and four present the classical Macaulay and the modern sparse resultant formulations. Section five presents Bezout's classical alternative to Euler's formulation for computing the resultant of two univariate polynomials, and Cayley's simple presentation of the same. Section six presents Dixon's generalization of the Bezout-Cayley formulation to the bivariate case, as well as our generalization of Dixon's method to the multivariate case of n variables. In section 6, we also prove why this generalization gives the *exact* resultant of a class of polynomial systems we call *generic, n-degree* systems. This generalization to n variables and Theorem 2.6.2 which proves that our method computes the exact resultant of a generic n-degree polynomial systems is new, and due to us. Finally in this section, we describe why it fails for arbitrary polynomial systems.

2.1 Definitions and Preliminaries

Let $X = \{x_1, \ldots, x_n\}$ be a set of n variables and $A = \{a_1, \ldots, a_m\}$, a set of m parameters distinct from X. Let $\mathcal{F} = \{p_1, \ldots, p_{n+1}\} \subset \mathbb{Q}[X, A]$ be a set of $n+1$ nonhomogeneous polynomials in X and A. Let $tdeg(p, X)$ be the total degree of the polynomial p in the variable set X and $deg(p, x_j)$, the degree of the polynomial p in the variable x_j. Let $\bar{\mathbb{Q}}$ be the algebraic closure of $\mathbb{Q}$.

The objective is to eliminate the variables X from $\mathcal{F}$ and obtain a polynomial purely in the parameters A whose vanishing is a necessary condition for the existence of solutions to $\mathcal{F}$. In other words, derive a polynomial in $\mathbb{Q}[A]$ which must vanish on all those particular values of the parameters $a_1, \ldots, a_m$ from $\bar{\mathbb{Q}}^m$ for which $\mathcal{F}$ has a common affine zero in $\bar{\mathbb{Q}}^n$. We will call any such polynomial a **Projection Operator**. The greatest common divisor of all possible projection operators is defined to be the **Resultant**. Let $\langle \mathcal{F} \rangle \subset \mathbb{Q}[X, A]$ be the ideal generated by $\mathcal{F}$ in the ring $\mathbb{Q}[X, A]$. Using the notation established so far in this section, the resultant and projection operators can be formally defined as follows:

Definition 2.1.1 *Polynomials in the set* $\langle \mathcal{F} \rangle \cap \mathbb{Q}[A]$ *are known as* **Projection Operators** *of* $\mathcal{F}$ *w.r.t.* X.

It can be proved that the set of projection operators forms an ideal in $\mathbb{Q}[A]$, and the resultant is then defined as follows:

Definition 2.1.2 *The generator of the ideal* $\langle \mathcal{F} \rangle \cap \mathbb{Q}[A]$ *is known as the* **Resultant** *of* $\mathcal{F}$ *w.r.t.* X.

The first thing to note is that when $\mathcal{F}$ is a non-homogeneous system, its projection operator, or even the resultant, is only a necessary condition for the existence of common zeros of $\mathcal{F}$, not sufficient. In fact, there may not even exist a sufficient condition in the nonhomogeneous case. However, the resultant is so defined that it *is* the *strongest* necessary condition. Projection operators are simply multiples of the resultant and their vanishing is also a necessary condition for $\mathcal{F}$ to have a common solution.

Another point to note is that for generic polynomials the resultant is always irreducible. However under specializations of A, the resultant may be reducible. A simple example is the resultant of $ax + b = 0$ and $bx + a = 0$ with respect to x is $(a - b)(a + b)$. Nevertheless, it follows from the definition of the resultant given in the previous paragraph that if a projection operator is irreducible, it must be the resultant.

Associated with the concept of a projection operator, is the concept of extraneous factors. All factors in a projection operator besides the resultant are **Extraneous Factors**. In other words, extraneous factors in a projection operator of a set of polynomials are all the factors of the ratio of the projection operator and the resultant of that set of polynomials. Now, if a polynomial system has solutions in $\bar{\mathbb{Q}}^n$ under all possible specializations of parameters A (i.e., the system $\mathcal{F}$ has an affine variety of dimension no less than m in $\bar{\mathbb{Q}}^{n+m}$), then the resultant must be identically zero. But if the dimension of the variety of $\mathcal{F}$ is less than m, then the resultant will be identically non-zero. In such a case, even though an identically zero polynomial trivially qualifies as a projection operator (since it is a multiple of the resultant with an extraneous factor of zero) it is obviously useless because it gives no information about the solutions of the polynomial system. Therefore for polynomial systems whose affine variety is of dimension $m - 1$ in $\bar{\mathbb{Q}}^{n+m}$, we are not only interested in computing projection operators, but in computing projection operators which are identically *non-zero*. In this thesis, we will qualify this a little bit by computing projection operators which are identically non-zero, at least when the system has a common solution in the *algebraic torus* $(\bar{\mathbb{Q}} - \{0\})^n$ for some specialization of the parameters. Of course, it is desirable that the number of extraneous factors in such projection operators also be minimized.

Lastly, for the sparse resultant formulation, as well as most of the results in this thesis, we will need to define the Newton polytope and related concepts. The *Newton Polytope* of a polynomial p is the convex hull of the set of exponents (treated as points in $\mathbb{R}^n$) of all terms in p when viewed as a polynomial in X. The *Minkowski sum* of two Newton polytopes Q and S is the set of all vector sums, $q+s$, $q \in Q$, $s \in S$.

2.2 Linearizing a Non-Linear Problem

Let $\mathcal{I} \subseteq \mathbb{Q}[X, A]$ be the ideal generated by the polynomial set $\mathcal{F}$. A straightforward way to compute the resultant of $\mathcal{F}$ is by computing the Gröbner basis of $\mathcal{I}$ using an elimination ordering which places the variables in X at a higher priority than parameters in A. The resultant is then an element of the basis. However, Gröbner basis methods are inefficient and consume a lot of space. Consider the following alternative approach.

Find another set of polynomials $\hat{\mathcal{F}} \subset \mathcal{I}$ with the following two properties:

1. Set of solutions of $\hat{\mathcal{F}}$ is preferably equal to or a superset (as small as possible) of the set of solutions of $\mathcal{F}$.

2. Number of different terms (in X) contained in the polynomials of $\hat{\mathcal{F}}$ is exactly as many as the number of polynomials in $\hat{\mathcal{F}}$.

If such an $\hat{\mathcal{F}}$ can be found, then it can be viewed as a linear system of homogeneous equations by treating each term in X (including the constant term) as an independent variable. A projection operator can then be found as the determinant of the coefficient matrix of this linear system. Clearly, this projection operator may have extraneous factors, and may be a multiple of the resultant.

The three most efficient methods for computing the resultant, viz. Macaulay, Dixon and sparse resultant formulations, try to find such $\hat{\mathcal{F}}$s. It turns out that in Macaulay and Sparse resultant formulations, each polynomial in $\hat{\mathcal{F}}$ is obtained simply by multiplying some polynomial in $\mathcal{F}$ by a monomial in the variables X. In the Dixon formulation however, the relationship between the polynomials of $\hat{\mathcal{F}}$ and $\mathcal{F}$ is much more complicated.

In this chapter we briefly outline these formulations and describe the nature and size of the set $\hat{\mathcal{F}}$ that each formulation constructs. Our objective is not to describe methods in full detail, rather to give just enough information so that the reader can get an idea of the resultant matrix sizes and limitations of these formulations. Before we do that, we give an example and its resultant. This example is used throughout this thesis to illustrate the applicability or limitations of different methods.

Example: (Resultant) Let $\mathcal{F} = \{p_1, p_2, p_3\} \subset \mathbb{Q}[a, b, c,\ x, y]$ where

$$p_1 \quad = \quad ax^2 + bxy + (b + c - a)x + ay + 3(c - 1),$$

$$
\begin{aligned}
p_2 &= 2a^2x^2 + 2abxy + aby + b^3, \\
p_3 &= 4(a-b)x + c(a+b)y + 4ab.
\end{aligned}
$$

The variables x and y are to be eliminated to obtain the resultant, a polynomial in the parameters a, b and c. From the Gröbner basis of this system, the resultant is found to be the following irreducible polynomial with 75 terms:

$$
\begin{aligned}
\rho(a,b,c) = {} & 144ca^2b^3 + 28a^2c^2b^5 - 144a^4b^4 + 36c^2a^3b^2 - 72c^3a^3b^2 + 4cb^9 + a^3b^6c^2 \\
& - 12c^2a^3b^5 - 240a^4b^2 - 264a^4c^3b + 336a^4c^2b - 144a^4cb - 72a^5cb - 384a^4cb^3 \\
& + 36a^5c^2 + 112a^5b^3 - 8ca^6b^2 - 80a^5b^4 - 32a^6b^2 + 20ca^4b^4 + 96ba^5 + 48a^3b^6 \\
& + 288a^4b^3 - 96a^3b^4 + 192a^3b^3 - 16a^2b^7 + 40a^3c^3b^3 - 192a^5b^2 + 16a^5b^4c \\
& + 16a^4b^5c - 8a^3b^6c - 24a^3c^2b^3 - 8a^2b^6c^2 + 2a^2c^2b^7 - 12a^2b^7c + 80a^3b^5 \\
& + 20ca^2b^6 - 264c^2a^2b^3 - 96a^6c^2b - 80a^5b^2c^2 + 320a^5cb^2 - 40a^5cb^3 - 16a^4c^3b^3 \\
& + 44a^4c^2b^3 - 216a^3cb^3 + 2a^3c^4b^3 + 16a^3cb^4 + 108a^3b^4c^2 - 20a^3b^5c - 28a^3c^3b^4 \\
& - 8a^2c^3b^5 + 2a^5c^2b^3 + 72a^4c^4b + 4a^2c^4b^4 + 36a^3c^4b^2 + 2ac^4b^5 - 24c^2a^2b^4 \\
& + 120c^3a^2b^3 + 64a^7b^2 + 72ca^2b^4 - 52c^2ab^6 + 96a^6cb - 4cab^7 + 2c^2ab^7 + 48cab^6 \\
& - 16a^2b^6 + ac^2b^8 + 4c^3ab^6 - 24a^5c^2b - 24a^4b^2c^2 + 216a^4cb^2 + 40a^4c^3b^2 - 48a^2b^4 \\
& + 36a^5c^4 - 72a^5c^3.
\end{aligned}
$$

Note that even though this resultant is irreducible, this may not be true for all resultants. $\square$

2.3 Macaulay Resultants

For $1 \leq i \leq n+1$, let $d_i = tdeg(p_i, X)$ and $d_M = 1 + \sum_1^{n+1}(d_i - 1)$. Let the set of terms T be $\{x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_n^{\alpha_n} \mid \alpha_1 + \alpha_2 + \cdots + \alpha_n \leq d_M\}$. To linearize the non-linear system $\mathcal{F}$ using the Macaulay formulation, each polynomial p_i in $\mathcal{F}$ is multiplied with certain terms to generate $\hat{\mathcal{F}}$ with $|T|$ equations in $|T|$ unknowns (which are terms in T). The coefficient matrix of $\hat{\mathcal{F}}$ so constructed is the **Macaulay matrix**. Columns of the Macaulay matrix are labeled by the terms in T in some order, and each row corresponds to a polynomial in $\mathcal{F}$ multiplied by a certain term.

The order in which polynomials are considered for selecting multipliers results in different (but equivalent) $\hat{\mathcal{F}}$s. Let M_σ denote the Macaulay matrix for a permutation σ of polynomials in $\mathcal{F}$. Macaulay also defined N_σ, a certain square submatrix of M_σ, such that the projection operator is the ratio of the determinants of M_σ and N_σ. See [32, 5, 27] for details of the construction and proofs of these properties.

Example: (Macaulay) For polynomial set in the example at the beginning of this section, $T = \{x^3, x^2y, x^2, xy^2, y^3, y^2, xy, x, y, 1\}$. $\hat{\mathcal{F}}$ is obtained by multiplying p_1 by x, y and 1, p_2 by x, y and 1 and p_3 by xy, x, y and 1. The Macaulay matrix, M_σ and N_σ are the 10×10 and 2×2 matrices in Figure 2.1.

Unfortunately, here both $det(M_\sigma)$ and $det(N_\sigma)$ are 0, so the resultant or a non-trivial projection operator cannot be computed directly. $\square$

$$
M_\sigma = \begin{array}{c} \\ x\times p_1 \\ y\times p_1 \\ p_1 \\ x\times p_2 \\ y\times p_2 \\ p_2 \\ xy\times p_3 \\ x\times p_3 \\ y\times p_3 \\ p_3 \end{array}
\begin{array}{c}
\begin{array}{cccccccccc} x^3 & x^2y & x^2 & xy^2 & y^3 & y^2 & xy & x & y & 1 \end{array} \\
\left(\begin{array}{cccccccccc}
a & b & b+c-a & 0 & 0 & 0 & a & 3(c-1) & 0 & 0 \\
0 & a & 0 & b & 0 & a & b+c-a & 0 & 3(c-1) & 0 \\
0 & 0 & a & 0 & 0 & 0 & b & b+c-a & a & 3(c-1) \\
2a^2 & 2ab & 0 & 0 & 0 & 0 & ab & b^3 & 0 & 0 \\
0 & 2a^2 & 0 & 2ab & 0 & ab & 0 & 0 & b^3 & 0 \\
0 & 0 & 2a^2 & 0 & 0 & 0 & 2ab & 0 & ab & b^3 \\
0 & 4(a-b) & 0 & c(a-b) & 0 & 0 & 4ab & 0 & 0 & 0 \\
0 & 0 & 4(a-b) & 0 & 0 & 0 & c(a-b) & 4ab & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & c(a-b) & 4(a-b) & 0 & 4ab & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 4(a-b) & c(a-b) & 4ab
\end{array}\right)
\end{array}
$$

$$
N_\sigma = \begin{array}{c} \\ p_1 \\ p_2 \end{array}
\begin{array}{c}
\begin{array}{cc} x^2 & y^2 \end{array} \\
\left(\begin{array}{cc} a & 0 \\ 2a^2 & 0 \end{array}\right)
\end{array}
$$

Figure 2.1: Matrices for Macaulay Formulation

Limitations of Macaulay Formulation

As is demonstrated in this example, this formulation may be unsuccessful in computing a nontrivial projection operator. One of the reasons is that $det(N_\sigma)$ can be identically zero, and the division cannot be carried out. Even if that is not the case, $det(M_\sigma)$ can be identically zero, hence giving only a trivial, identically zero projection operator. Also, under coefficient specializations, this formulation may give rise to extraneous factors.

2.4 Sparse Resultants

The Sparse resultant approach is based on the recent results pertaining to sparse polynomial systems (Bernstein [3], Gelfand et. al. [23] and Sturmfels [40]). Sparse resultants first appeared in Sturmfels & Zelevinsky [41], its matrix formulation was given by Canny & Emiris [6] and was successively improved in [7, 15, 16] using better heuristics. The Macaulay formulation is based on the Bezout bound on the number of solutions to a set of polynomials, which is quite loose. Instead, the sparse formulation uses the recently developed, and significantly tighter, *BKK bound* [23] to construct a smaller resultant matrix than Macaulay.

Let $Q_i \subset \mathbb{R}^n$ be the Newton polytope (w.r.t. X) of the polynomial p_i in $\mathcal{F}$. Let $Q = Q_1 + \cdots + Q_{n+1} \subset \mathbb{R}^n$ be the Minkowski sum of Newton polytopes of all the polynomials in $\mathcal{F}$. Let $\mathcal{E}$ be the set of exponents (lattice points of $\mathbb{Z}^n$) in Q obtained after applying a small perturbation to Q to move as many boundary lattice points as possible outside Q.

Construction of $\hat{\mathcal{F}}$ in the Sparse resultant formulation is similar to the Macaulay formulation – each polynomial, p_i in $\mathcal{F}$ is multiplied with certain terms to generate $\hat{\mathcal{F}}$ with $|\mathcal{E}|$ equations in $|\mathcal{E}|$ unknowns (which are terms in $\mathcal{E}$), and its coefficient matrix is the **sparse resultant matrix**. A projection operator for $\mathcal{F}$ is simply the determinant of this matrix. Columns of the sparse resultant matrix are labeled by the terms in $\mathcal{E}$ in some order, and each row corresponds to a polynomial in $\mathcal{F}$ multiplied by a certain term.

Recall that the size of the Macaulay matrix is $|T|$. Size of the sparse resultant matrix, $|\mathcal{E}|$, is typically smaller than $|T|$, especially when the BKK bound of $\mathcal{F}$ is tighter than the Bezout bound. Algorithms in [15, 16, 7] construct matrices using some greedy heuristics which may result in smaller matrices, but in the worst case, the size can still be $|\mathcal{E}|$.

Example: (Sparse) Continuing with the same example, construction of the Newton polytopes and their Minkowski sum (after a perturbation in the positive direction)

$$
\begin{array}{cc}
 & \begin{array}{ccccccccc} xy & xy^2 & xy^3 & x^2y & x^2y^2 & x^2y^3 & x^3y & x^3y^2 & x^4y \end{array} \\
\begin{array}{c} xy \times p_1 \\ xy^2 \times p_1 \\ x^2y \times p_1 \\ xy \times p_2 \\ xy^2 \times p_2 \\ x^2y \times p_2 \\ xy \times p_3 \\ xy^2 \times p_3 \\ x^2y^2 \times p_3 \end{array} &
\left(\begin{array}{ccccccccc}
3(c-1) & a & 0 & b-a+c & b & 0 & a & 0 & 0 \\
0 & 3(c-1) & a & 0 & b-a+c & b & 0 & a & 0 \\
0 & 0 & 0 & 3(c-1) & a & 0 & b-a+c & b & a \\
b^3 & ab & 0 & 0 & 2ab & 0 & 2a^2 & 0 & 0 \\
0 & b^3 & ab & 0 & 0 & 2ab & 0 & 2a^2 & 0 \\
0 & 0 & 0 & b^3 & ab & 0 & 0 & 2ab & 2a^2 \\
4ab & c(a+b) & 0 & 4(a-b) & 0 & 0 & 0 & 0 & 0 \\
0 & 4ab & c(a+b) & 0 & 4(a-b) & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 4ab & c(a+b) & 0 & 4(a-b) & 0
\end{array}\right)
\end{array}
$$

Figure 2.2: Sparse Resultant Matrix

reveals that $\mathcal{E} = \{xy, xy^2, xy^3, x^2y, x^2y^2, x^2y^3, x^3y, x^3y^2, x^4y\}$. The 9×9 sparse resultant matrix is given in Figure 2.2.

The determinant of the sparse resultant matrix is the following projection operator with 205 terms (in expanded form):

$$\varrho = -2a^2\,(a-b-c)\left(a^2c + abc - 4ab + 4b^2\right)\rho\,(a,b,c)\,.$$

Notice that there are 3 extraneous factors. □

Limitation of Sparse Resultant Formulation

Even though, in this particular example we are able to obtain a nontrivial projection operator, it may not always be the case. Although it seems rare, the sparse resultant matrix can also be singular for certain polynomial systems (Example 2 in Table 6.2), in which case the determinant will be the trivial, identically zero, projection operator. Also note that extraneous factors appear in the result of this formulation also.

2.5 Bezout-Cayley Formulation

Bezout gave a method which computes the resultant of two polynomials with respect to a single variable by constructing a resultant matrix which is smaller than the Euler-Sylvester matrix. Bezout's approach was quite complicated, and Cayley gave a neat and simple reformulation of Bezout's method. The formulation we present in this section was given by Cayley.

Consider a set $\mathcal{F}$ of two univariate polynomials $\{p_1(x_1), p_2(x_1)\}$. Let

$$d_{max} = max\left(deg\left(p_1, x_1\right), deg\left(p_2, x_1\right)\right).$$

Consider the polynomial

$$\Delta(x_1, \bar{x}_1) = \begin{vmatrix} p_1(x_1) & p_2(x_1) \\ p_1(\bar{x}_1) & p_2(\bar{x}_1) \end{vmatrix},$$

where $\bar{x}_1$ is a new variable and $p_1(\bar{x}_1)$ stands for uniformly replacing x_1 by $\bar{x}_1$ in p_1. Substituting $\bar{x}_1$ for x_1 in Δ would make it vanish, which means that $x_1 - \bar{x}_1$ fully divides Δ. Let,

$$\delta(x_1, \bar{x}_1) = \frac{\Delta(x_1, \bar{x}_1)}{(x_1 - \bar{x}_1)} = \frac{p_1(x_1)p_2(\bar{x}_1) - p_2(x_1)p_1(\bar{x}_1)}{(x_1 - \bar{x}_1)}.$$

The polynomial δ is known as the *Dixon polynomial.* It is a $d_{max} - 1$ degree polynomial in $\bar{x}_1$, and is symmetric in x_1 and $\bar{x}_1$. Now, every common zero of $p_1(x_1)$ and $p_2(x_1)$ is a zero of $\delta(x_1, \bar{x}_1)$, no matter what value $\bar{x}_1$ has. Thus at a common zero of p_1 and p_2, coefficients of all powers of $\bar{x}_1$ in $\delta(x_1, \bar{x}_1)$ must vanish. These coefficients of the power products of $\bar{x}$ in the Dixon polynomial form our linear system of equations $\hat{\mathcal{F}}$ for the Bezout-Cayley formulation. It has d_{max} equations corresponding to the coefficients of all the powers of $\bar{x}_1$ (viz. $\bar{x}_1^i$ for each $0 \leq i \leq d_{max} - 1$), each of which is a $d_{max} - 1$ degree polynomial in x_1. So, if D is the $d_{max} \times d_{max}$ coefficient matrix of $\hat{\mathcal{F}}$, then the linear system

$$\hat{\mathcal{F}} \equiv D \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_{d_{max}-1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

where $v_i = x_1^i$.

If a common affine zero exists for $\mathcal{F}$ (say $x_1 = c_1$), then this results in a nontrivial solution for $\hat{\mathcal{F}}$ also, viz., $v_0 = 1, v_1 = c_1, v_2 = c_1^2, \ldots, v_{d_{max}-1} = c_1^{d_{max}-1}$. Hence if $\mathcal{F}$ has a common affine zero, then $\hat{\mathcal{F}}$ has a nontrivial solution, implying that the determinant of D is zero. This gives a necessary condition on the coefficients of p_1 and p_2 for them to have a common affine zero. So, D is the resultant matrix here.

2.6 The Classical Dixon Formulation

Dixon extended Bezout and Cayley's approach for univariate polynomials to systems with more than one variables. Although Dixon explicitly generalized Cayley's method presented in the previous section only to the two variable case in [12], it can be generalized to any number of variables. In this section, we present such a generalization to an arbitrary number of variables, and prove that the generalization successfully computes the exact resultant of a class of polynomials known as *Generic n-Degree* polynomials. The generalization and the theorem which proves that it computes the exact resultant of a generic n-degree polynomial systems is new and due to us.

2.6.1 Dixon's Generalization of the Bezout-Cayley Formulation

Recall that $\mathcal{F} = \{p_1(x_1, \cdots, x_n), \cdots, p_{n+1}(x_1, \cdots, x_n)\}$ is the set of $n+1$ polynomials in n variables, with coefficients from $\mathbb{Q}[a_1, \ldots, a_m]$. We form an $(n+1) \times (n+1)$ determinant similar to the determinant in the last section. Let this determinant be defined as follows:

$$\Delta(x_1, \cdots, x_n, \bar{x}_1, \cdots, \bar{x}_n) = \begin{vmatrix} p_1(x_1, x_2, \cdots, x_n) & \cdots & p_{n+1}(x_1, x_2, \cdots, x_n) \\ p_1(\bar{x}_1, x_2, \cdots, x_n) & \cdots & p_{n+1}(\bar{x}_1, x_2, \cdots, x_n) \\ p_1(\bar{x}_1, \bar{x}_2, \cdots, x_n) & \cdots & p_{n+1}(\bar{x}_1, \bar{x}_2, \cdots, x_n) \\ \cdots & \cdots & \cdots \\ p_1(\bar{x}_1, \bar{x}_2, \cdots, \bar{x}_n) & \cdots & p_{n+1}(\bar{x}_1, \bar{x}_2, \cdots, \bar{x}_n) \end{vmatrix},$$

where $\bar{x}_1, \cdots, \bar{x}_n$ are new variables and $p_i(\bar{x}_1, \cdots, \bar{x}_k, x_{k+1}, \ldots, x_n)$ stands for uniformly replacing x_j by $\bar{x}_j$ for $1 \leq j \leq k \leq n$ in p_i. This matrix is known as the **Cancellation matrix**.

Since $x_i = \bar{x}_i$, for each $1 \leq i \leq n$, is a zero of Δ, Δ can be divided by $\prod_{i=1}^{n}(x_i - \bar{x}_i)$. Let,

$$\delta(x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n) = \frac{\Delta(x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n)}{(x_1 - \bar{x}_1) \cdots (x_n - \bar{x}_n)}.$$

The polynomial δ is known as the **Dixon Polynomial**. Again, any common zero of $\mathcal{F}$ makes the Dixon polynomial vanish, no matter what the values of $\bar{x}_1, \cdots, \bar{x}_n$. This is because at such a zero, the first row of Δ is 0. Therefore the coefficients of the various power products of $\bar{x}_1, \ldots, \bar{x}_n$ in the Dixon polynomial, which are polynomials in $x_1, \ldots, x_n$, must also vanish at all common zeros of $\mathcal{F}$. Our set $\hat{\mathcal{F}}$ in the case of Dixon formulation is then defined to be this set of all the polynomials in $x_1, \ldots, x_n$ which are coefficients of the power products of $\bar{x}_1, \ldots, \bar{x}_n$ in the Dixon polynomial. The coefficient matrix of $\hat{\mathcal{F}}$ (when viewed as linear equations in power products of $x_1, \ldots, x_n$), is known as the Dixon matrix. Some properties and crude bounds on the size of the Dixon matrix can be derived as follows:

Let,

$$d_{max_i} = max(deg(p_1, x_i), \ldots, deg(p_{n+1}, x_i)),$$

for all $1 \leq i \leq n$. It is easy to see by expansion that the determinant of the Cancellation matrix is a polynomial of degree at most $(n+1-i) \times d_{max_i}$ in $\bar{x}_i$ and at most $i \times d_{max_i}$ in x_i for all $1 \leq i \leq n$. Since the Dixon polynomial is obtained by dividing the determinant of Cancellation matrix by $\prod_{i=1}^{n}(x_i - \bar{x}_i)$, the Dixon polynomial cannot be of degree greater than $((n+1-i) \times d_{max_i}) - 1$ in $\bar{x}_i$ and $(i \times d_{max_i}) - 1$ in x_i. Consequently, $\hat{\mathcal{F}}$ has at most $s = \prod_{i=1}^{n}((n+1-i) \times d_{max_i}) = n! \times \prod_{i=1}^{n} d_{max_i}$ equations (one for each power product of $\bar{x}_1, \ldots, \bar{x}_n$), each of which is of degree no more than $(i \times d_{max_i}) - 1$ in x_i. Thus, there are at most $\prod_{i=1}^{n} i \times d_{max_i} = n! \times \prod_{i=1}^{n} d_{max_i} = s$ power products in $x_1, \ldots, x_n$ in the equations of $\hat{\mathcal{F}}$. Therefore the Dixon matrix, D, of $\mathcal{F}$ which is the coefficient matrix of $\hat{\mathcal{F}}$, is no larger than $s \times s$, where $s = n! \prod_{i=1}^{n} d_{max_i}$.

We now discuss a specific case in which the determinant of D is exactly the resultant of $\mathcal{F}$.

The Generic N-Degree Case

The set $\mathcal{F}$ is called *generic n-degree* if there exists a set of n integers, $k_1, \cdots, k_n$ such that all polynomials, $p_1, \cdots, p_{n+1}$, are of the kind:

$$p_j = \Sigma_{i_1=0}^{k_1} \cdots \Sigma_{i_n=0}^{k_n} a_{j,i_1,\ldots,i_n} x_1^{i_1} \cdots x_n^{i_n}, \ 1 \leq j \leq n+1,$$

where each a is a distinct indeterminate. Now, consider the system $\mathcal{F}_n$ consisting only of $\{p_1, \ldots, p_n\}$. Since $\mathcal{F}_n$ is a generic system, all its roots will be distinct. It is also easy to see that since this is an n-degree system, it will have at least $n! \prod_{i=1}^n k_i$ roots. For example, if all the coefficients, a, are distinct and independent, then the following n-degree system

$$p_j = \prod_{i=1}^{n} \left(\Sigma_{l=0}^{k_i} a_{j,i,l} x_i^l \right), \quad 1 \leq j \leq n,$$

clearly has $n! \prod_{i=1}^n k_i$ distinct roots. Therefore, we have shown that the generic n-degree system $\mathcal{F}_n$ has at least $n! \prod_{i=1}^n k_i$ distinct roots. The resultant of $\mathcal{F}$ is then the product of the value of p_{n+1} at each of these roots. Clearly, this resultant must at least be of degree $n! \prod_{i=1}^n k_i$ in the coefficients of p_{n+1}, viz. $a_{n+1,i_1,\ldots,i_n}$. By symmetry of a generic n-degree system, we have proved the following lemma:

Lemma 2.6.1 *The resultant of a generic n-degree system of n-degree* $(k_1, \cdots, k_n)$ *is of total degree no less than* $(n+1)! \prod_{i=1}^n k_i$ *in the coefficients of its polynomials.*

Now we will derive four properties about the Dixon matrix of a generic n-degree system, which, together with the above lemma, will help us prove that the determinant of the Dixon matrix of a generic n-degree system is *exactly* the resultant. These four properties are:

1. The Dixon matrix of a generic n-degree system is square

2. The Dixon matrix of a generic n-degree system is non-singular.

3. The total degree of the determinant of the Dixon matrix of a generic n-degree system in its coefficients is no more than $(n+1)! \prod_{i=1}^{n} k_i$.

4. The determinant of the Dixon matrix of a generic n-degree system is its projection operator.

If these four properties can be proven, then by (4), the resultant must divide the determinant of the Dixon matrix. But since, by (3), the total degree of the determinant in the coefficients of the generic n-degree system is *no more than* $(n+1)! \prod_{i=1}^{n} k_i$, and the total degree of the resultant, by previous lemma, is *no less than* $(n+1)! \prod_{i=1}^{n} k_i$, it follows that the resultant and the determinant of the Dixon matrix are equivalent, up to a constant factor (which *cannot* be zero, by (2)). Thereby proving the theorem:

Theorem 2.6.2 *The determinant of the Dixon matrix of a generic n-degree system is exactly its resultant (i.e., does not have any extraneous factors).*

Proof: We only need to prove the four properties about the Dixon matrix of a generic n-degree system enumerated above.

1. From the properties of the Dixon matrix proven in the previous subsection, the number of rows and the number of columns in the Dixon matrix of a generic n-degree system cannot exceed $n!\prod_{i=1}^{n} d_{max_i} = n!\prod_{i=1}^{n} k_i$. If we can show an instance of the generic n-degree system for which the number of rows and the number of columns equal exactly $n!\prod_{i=1}^{n} k_i$, then we would have shown that:

   > The Dixon matrix of any generic n-degree system is square and of size $(n!\prod_{i=1}^{n} k_i) \times (n!\prod_{i=1}^{n} k_i)$.

   One such instance is the following:

$$\begin{aligned} p_j &= \prod_{i=1}^{j-1}\left(x_i^{k_i}+1\right)\left(x_j^{k_j}+a_j\right)\prod_{i=j+1}^{n}\left(x_i^{k_i}+1\right) \qquad 1 \le j \le n, \\ p_{n+1} &= \prod_{i=1}^{n}\left(x_i^{k_i}+1\right), \end{aligned}$$

   where all a_j are independent parameters. Clearly this is a specific instance of a system of n-degree $\langle k_1, \ldots, k_n\rangle$. By examining its cancellation matrix, it can be easily shown that its Dixon polynomial is:

$$\delta = c\prod_{i=1}^{n}\left(x_i^{k_i}+1\right)^{i-1}\left(\bar{x}_i^{k_i}+1\right)^{n-i}\left(\sum_{j=1}^{k_i} x_i^{j-1}\bar{x}_i^{k_i-j}\right),$$

   where c is a constant (a function of the a_j). It follows by simple algebra that when this Dixon polynomial is viewed as a polynomial in X, it has $n!\prod_{i=1}^{n} k_i$ terms, and when it is viewed as a polynomial in $\bar{X}$, it again has $n!\prod_{i=1}^{n} k_i$ terms.

This proves that the number of rows and the number of columns in its Dixon matrix are exactly $n!\prod_{i=1}^{n} k_i$. As we mentioned earlier, this proves that The Dixon matrix of a generic n-degree is square and of size $(n!\prod_{i=1}^{n} k_i)\times(n!\prod_{i=1}^{n} k_i)$.

2. To prove that the Dixon matrix is non-singular, we only need to show that there exists an instance of any generic n-degree system whose Dixon matrix is non-singular and exactly of the size in the previous item, then we would have shown that:

   The Dixon matrix of any generic n-degree is non-singular.

   One such instance is the following:

$$p_j = \prod_{i=1}^{n} \left(x_i^{k_i} - a_j\right) \qquad for 1 \leq j \leq (n+1),$$

   where all a_j are independent parameters. Clearly this is a specific instance of a system of n-degree $\langle k_1, \ldots, k_n \rangle$. By examining its cancellation matrix, it can be easily shown that its Dixon polynomial is:

$$\delta = c \prod_{i=1}^{n} \frac{x_i^{k_i} - \bar{x}_i^{k_i}}{x_i - \bar{x}_i} \prod_{j=1}^{i-1} \left(x_i^{k_i} - \bar{x}_j^{k_j}\right),$$

   where c is a constant (a function of the a_j). By expansion, it can be shown that the Dixon matrix constructed from the above dixon polynomial will be square, of size $n!\prod_{i=1}^{n} k_i \times n!\prod_{i=1}^{n} k_i$ and non-singular. In fact, it can be shown that the determinant of such a matrix is:

$$c^{n!\prod_{i=1}^{n} k_i}.$$

In fact, by expansion of the cancellation matrix, it can also be shown that

$$c = \prod_{i=1}^{n} \prod_{j=i+1}^{n} (a_i - a_j),$$

thus proving that $c \neq 0$, and that the Dixon matrix of this specialized system is indeed non-singular.

3. It can be easily confirmed by examining the Cancellation matrix that the coefficients of the Dixon polynomial (when viewed as a polynomial in $X \cup \bar{X}$) have total degree no more than $n+1$ in the coefficients of a generic n-degree $\mathcal{F}$. Since these coefficients form the entries of the Dixon matrix, it follows that no entry in the Dixon matrix is of degree more than $n+1$ in the coefficients of $\mathcal{F}$. In the previous item we proved that the Dixon matrix of a generic n-degree system is $n!\prod_{i=1}^{n} k_i$. Therefore, it follows that the determinant of the Dixon matrix does not have a total degree larger than $(n+1)!\prod_{i=1}^{n} k_i$ in the coefficients of $\mathcal{F}$.

4. It can easily be confirmed by examining the Cancellation matrix that the constant term in the Dixon polynomial of a generic n-degree system is the $(n+1) \times (n+1)$ determinant:

$$\begin{vmatrix} a_{1,0,0,0,\ldots,0,0} & a_{1,1,0,0,\ldots,0,0} & a_{1,0,1,0,\ldots,0,0} & \cdots & a_{1,0,0,0,\ldots,0,1} \\ a_{2,0,0,0,\ldots,0,0} & a_{1,1,0,0,\ldots,0,0} & a_{1,0,1,0,\ldots,0,0} & \cdots & a_{1,0,0,0,\ldots,0,1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n+1,0,0,0,\ldots,0,0} & a_{n+1,1,0,0,\ldots,0,0} & a_{n+1,0,1,0,\ldots,0,0} & \cdots & a_{n+1,0,0,0,\ldots,0,1} \end{vmatrix}$$

Now, let the set of s monomials labelling the columns of the Dixon matrix be $v_0 = 1, v_1, v_2, \cdots, v_s$, where v_0 is the constant term, which we have just shown to have a non-zero coefficient in the Dixon polynomial, and v_i for $i \geq 1$ are those power products in X which have a non-zero coefficient in the Dixon polynomial, and whose total degree in X is greater than 1. Now, all common affine zeros of $\mathcal{F}$ must give a solution for $\hat{\mathcal{F}}$ in which $v_0 = 1$. In other words, if $\mathcal{F}$ has a common affine zero, then $\hat{\mathcal{F}}$ has a nontrivial solution, implying that the determinant of the Dixon matrix D (which is the coefficient matrix of $\hat{\mathcal{F}}$, and already shown to be square) must vanish at all zeros of $\mathcal{F}$. Therefore the determinant of the Dixon matrix of $p_1, \cdots, p_{n+1}$ must vanish for them to have a common zero. In other words, the determinant of the Dixon matrix of a generic n-degree system is a projection operator of $\mathcal{F}$.

From these four properties, the theorem follows. $\square$

Therefore, we have shown that the determinant of the Dixon matrix of a generic n-degree polynomial system is exactly its resultant. However, serious problems can arise when the polynomials are not generic or not n-degree, as demonstrated by the following example:

Example: (Dixon) The Dixon matrix for the previous example is of dimension 2×3 and is given in Figure 2.3. Since the Dixon matrix is rectangular, its determinant cannot be computed. Hence the resultant cannot be computed by this formulation either. $\square$

As this example illustrates, in the non-generic case or when the polynomials

$$
\begin{array}{cc}
 & \begin{array}{ccc} \quad\quad\quad\quad 1 \quad\quad\quad\quad & \quad\quad\quad x \quad\quad\quad & \quad\quad\quad y \quad\quad\quad \end{array} \\
\begin{array}{c} \bar{x} \\ \\ \\ \\ \\ 1 \end{array} & \left(\begin{array}{lll}
\begin{array}{l} 4ca^2b^2 - c^2b^4 + 12cab^2 \\ -12ca^2b + 12a^2b - 12ab^2 \\ -c^2b^3a + cb^3a^2 - 4ab^4 \\ -cb^5 - 4a^3b^2 + 8a^2b^3 \end{array} & \begin{array}{l} 4a^3b^2 - 8a^4b \\ -6a^3c + 6a^3c^2 \\ +6a^2c^2b - 6ca^2b \\ -cb^3a^2 - cb^4a \end{array} & \begin{array}{l} 6a^2c^2b - 6cab^2 \\ +4a^2b^3 - 8a^3b^2 \\ -cb^5 - cb^4a \\ -6ca^2b + 6ac^2b^2 \end{array} \\
\begin{array}{l} 8ca^2b^2 - 4b^5 + 24cab^2 \\ -30ca^2b + 24a^2b - 24ab^2 \\ -6a^3c + 6a^3c^2 - cb^3a^2 \\ -8a^4b - 4a^3b^2 - cb^4a \\ +8a^2b^3 + 6a^2c^2b + 4ab^4 \end{array} & \begin{array}{l} 12a^3b - 4a^2b^2 \\ -8a^4 + 2a^3c^2 \\ +2ca^2b^2 - 2a^4c \\ +2a^2c^2b \end{array} & \begin{array}{l} 2ac^2b^2 + 12a^2b^2 \\ +2a^2c^2b - 2a^3cb \\ -8a^3b - 4b^3a \\ +2cb^3a \end{array}
\end{array}\right)
\end{array}
$$

Figure 2.3: Dixon Matrix

are not n-degree, the Dixon matrix may be rectangular, or even if it is square, it may be singular. That is why Dixon formulation was considered very restrictive and abandoned. We discuss this case next.

2.6.2 The Case of Arbitrary Polynomials

Consider the following three polynomials in two variables.

$$
\begin{aligned}
p_1 &= x^2 + axy - y + b \\
p_2 &= xy + (a+b)y - c \\
p_3 &= bx + y + 2ac.
\end{aligned}
$$

Here a, b and c are the parameters and x and y are the variables to be eliminated.

This set of polynomials is clearly not generic (because for example, the coefficient of x^2 in p_1, 1, is not an independent parameter, and the coefficient of y in p_2 is not independent since it depends on the coefficients of p_1). This set is also not n-degree (because for example, even though the maximum degree of x and y is two and one respectively, there is no term x^2y in any of the polynomials). So what must one do in this case? The analysis in the previous section was developed for generic n-degree polynomials, so it may not work directly for this set of polynomials. The Dixon matrix may be rectangular, or even if it is square, it may be singular.

Sometimes one possible way to deal with such a situation is as follows. Write down the generic polynomials of n-degree (2,1) in x and y:

$$\begin{aligned}
p_1' &= a_1x^2y + a_2x^2 + a_3xy + a_4x + a_5y + a_6 \\
p_2' &= a_7x^2y + a_8x^2 + a_9xy + a_{10}x + a_{11}y + a_{12} \\
p_3' &= a_{13}x^2y + a_{14}x^2 + a_{15}xy + a_{16}x + a_{17}y + a_{18}
\end{aligned}$$

Now construct its 4×4 Dixon matrix D. Compute its determinant $det(D)$ and then substitute the values $a_1 = 0, a_2 = 1, a_3 = a, a_4 = 0, a_5 = -1, \cdots, a_{18} = 2ac$ into $det(D)$. This would give a polynomial in a, b and c (call it p_{dix}). The polynomial p_{dix} vanishes at all those values of a, b and c at which p_1, p_2 and p_3 has an affine zero. Hence, p_{dix} is a projection operator of $\mathcal{F}$.

This approach could be followed for any arbitrary set of polynomials $\mathcal{F}$ as follows:

1. Construct generic n-degree polynomials $\mathcal{F}'$ which when specialized in a certain way (say a mapping ψ from the parameters of $\mathcal{F}'$ to the parameters of $\mathcal{F}$), becomes $\mathcal{F}$.

2. Find the Dixon resultant of $\mathcal{F}'$.

3. Specialize the parameters in this Dixon resultant polynomial using ψ to get the polynomial p_{dix}.

However, most of the times, when this approach is followed, p_{dix} becomes identically zero. This is something which was avoided while working with generic n-degree polynomials. More importantly, this approach is also very inefficient because $\mathcal{F}'$ typically has much larger polynomials which are also of a much larger degree than $\mathcal{F}$. In fact, adding terms and making the polynomials generic amounts to destroying all sparsity in the input polynomials and dealing directly with the dense case.

In this thesis, we have adopted a different approach. We remove the restriction of generic n-degree from the construction of the previous section. So now, we are given a set of any arbitrary polynomials. Construct Δ as in the last subsection. Divide its determinant by $(x_1 - \bar{x}_1)\cdots(x_n - \bar{x}_n)$ to get the Dixon polynomial δ. This polynomial now may have a degree less than or equal to $((n+1-i)\times d_{max_i}) - 1$ in each $\bar{x}_i$ and less than or equal to $(i \times d_{max_i}) - 1$ in each x_i for all $1 \leq i \leq n$. Construct the set of equations $\hat{\mathcal{F}}$ from the Dixon polynomial as before, except that now $\hat{\mathcal{F}}$ may have less than or equal to s equations in less than or equal to s variables. We continue to call the coefficient matrix of $\hat{\mathcal{F}}$ to be the **Dixon matrix** D *which now may be an* $s_1 \times s_2$ *matrix where* $s_1 \leq s$ *and* $s_2 \leq s$.

Notice though that the determinant of the Dixon matrix may not even give us a necessary condition for the existence of affine zeros of $\mathcal{F}$. In the generic, n-degree case, we were able to prove that the column corresponding to $x_1^0 \cdots x_n^0$ would always be a non-zero vector, thus enabling us to prove that $\hat{\mathcal{F}}$ has a non-trivial solution at all affine zeros of $\mathcal{F}$. But the Dixon matrix of arbitrary systems may not contain the column corresponding to the monomial $x_1^0 \cdots x_n^0$ hence making it possible for $\hat{\mathcal{F}}$ to have only trivial solutions at all affine zeros of $\mathcal{F}$. In such a case, obviously the determinant of the Dixon matrix need not vanish at all common affine zeros of $\mathcal{F}$.

Even if the Dixon matrix *does* contain the column corresponding to $x_1^0 \cdots x_n^0$, the Dixon matrix could be singular, giving only a trivial projection operator. Worst of all, the Dixon matrix could be a rectangular matrix (because s_1 and s_2 may not be same), in which case, there does not even exist the possibility of computing the determinant.

We deal with all of these possibilities together and extend the applicability of the Dixon formulation in the next chapter.

Chapter 3

The Extended Dixon Formulation

In this chapter we develop an approach which uses rank submatrix computation, *RSC*, to extend Dixon's generalization of the Bezout-Cayley formulation to a larger class of polynomial systems. This leads to a general method for extracting a non-trivial multiple of the resultant from degenerate resultant matrices in any resultant formulation. This extension of Dixon's generalization is our topic of discussion in the first section. In the second section, we first describe a traditional method which uses perturbation of the input system to deal with degeneracies in resultant matrices, and then explain how the *RSC* method offers an efficient alternative to this inefficient traditional approach – *even* for the Macaulay and sparse resultant formulations. We show how the perturbation technique destroys the inherent sparsity in the input system and increases the complexity of the problem, while in contrast, our *RSC* method retains all of the sparsity and efficiency in the input system. In the last section we derive some bounds on our extension to the Dixon formulation using only the concepts which have been discussed so far.

3.1 Rank Submatrix Computations (RSC)

Let $\mathcal{C}$ be a constraint on the variables $X = \{x_1, \ldots, x_n\}$ of the form $x_{i_1} \neq 0 \wedge \cdots \wedge x_{i_k} \neq 0$, where $1 \leq i_j \leq n$. Recall that $A = \{a_1, \ldots, a_m\}$ is the set of parameters. Now, given a set $\mathcal{F} \subset \mathcal{Q}[\mathcal{X}, \mathcal{A}]$ of $n+1$ polynomials, $\varrho_{\mathcal{C}}$, called the **projection operator of $\mathcal{F}$ modulo $\mathcal{C}$** is defined to be any polynomial in $\mathbb{Q}[a_1, \ldots, a_m]$ which vanishes for all those particular values of parameters A from $\bar{\mathbb{Q}}^m$ for which $\mathcal{F}$ has a common affine zero in $\bar{\mathbb{Q}}^n$ which also *satisfies the set of constraints* $\mathcal{C}$. Again, since identically zero projection operators are useless, we are only interested in computing $\varrho_{\mathcal{C}}$s which are identically non-zero.

This is a generalization of the problem as posed towards the end of the previous chapter. For instance, if one is interested in a projection operator, ϱ, as defined in Definition 2.1.1, then $\mathcal{C}$ can be the empty set Φ, and $\varrho = \varrho_{\Phi}$. On the other hand, if $\mathcal{C} = (x_1 \neq 0) \wedge \cdots \wedge (x_n \neq 0)$, then it amounts to computing a projection operator which must vanish on all those parameter specializations for which $\mathcal{F}$ has a common zero in the *algebraic torus*, $(\bar{\mathbb{Q}} - 0)^n$. The problem of computing projection operators modulo a set of constraints on the variables is in itself also important because in areas such as geometric theorem proving, one occasionally encounters constraints under which solutions are sought.

Let the Dixon matrix of $\mathcal{F}$ be of dimension $s_1 \times s_2$ where $s_1 \leq n!\Pi_{i=1}^{n} d_{max_i}$ is the number of rows (corresponding to the coefficients of all power products of $\bar{x}$s in the Dixon polynomial) and $s_2 \leq n!\Pi_{i=1}^{n} d_{max_i}$ is the number of columns (corresponding to all power products of xs in the Dixon polynomial). Let the columns of the Dixon matrix be denoted by m_i (m_1 being the first column of the Dixon matrix and so on). Given a set of constraints $\mathcal{C}$ as above, let $nvcol(\mathcal{C})$ denote the set of all columns which are labeled by monomials which are guaranteed not to vanish at points satisfying the constraint $\mathcal{C}$. So, if $\mathcal{C} = \{x_2 \neq 0, x_5 \neq 0\}$, then the column labeled by $x_2^3 x_5^2$ would be in $nvcol(\mathcal{C})$, as would columns labeled by 1 or any of x_2^i or x_5^j. If $\mathcal{C} = \Phi$, then only the column labeled by 1 will be in $nvcol(\mathcal{C})$. If $\mathcal{C} = (x_1 \neq 0) \wedge \cdots \wedge (x_n \neq 0)$, then all columns of the Dixon matrix will be in $nvcol(\mathcal{C})$.

First we prove a lemma about polynomials all of whose coefficients are from the algebraic closure of the field of rational numbers $\bar{\mathbb{Q}}$. Let $\mathcal{G} \subset \bar{\mathbb{Q}}[x_1, \cdots, x_n]$ be a set of $n+1$ polynomials. In other words, no parameter a_i appears in $\mathcal{G}$ that is, all its coefficients are elements of $\bar{\mathbb{Q}}$, instead of $\bar{\mathbb{Q}}[A]$. Also, as before, let $\mathcal{C}$ be a set of constraints on the variables X. Let N be the $s_1 \times s_2$ Dixon matrix of $\mathcal{G}$. Also let

$$\begin{aligned}\mathcal{N}_1 \;=\; \{Z \mid & \text{ Z is an } \; s_1 \times (s_2 - 1) \; \text{ submatrix of } \; N \; \text{ obtained by deleting a} \\ & \text{column which belongs to } \; nvcol(\mathcal{C}) \; \text{ from } \; N\}\end{aligned}$$

Note that if $\mathcal{C}$ is $x_1 \neq 0 \wedge \cdots \wedge x_n \neq 0$ then $\mathcal{N}_1$ contains all the $s_1 \times (s_2 - 1)$ submatrices of N. Now we prove a lemma relating the rank of the Dixon matrix N to the ranks of all the matrices which are elements of $\mathcal{N}_1$.

Simply stated, this lemma means that if $\mathcal{G}$ has a common zero, then the rank of its Dixon matrix does not change upon removing from it any column which is labeled by a monomial which cannot vanish at any point satisfying $\mathcal{C}$ (i.e., the column is in $nvcol(\mathcal{C})$). Formally, the statement is as follows:

Lemma 3.1.1 *If $\mathcal{G}$ has a common affine zero in $\bar{\mathbb{Q}}^n$ which satisfies $\mathcal{C}$, then*

$$\forall Z \epsilon \mathcal{N}_1, rank(Z) = rank(N)$$

Proof : Let $x_1 = c_1, \cdots, x_n = c_n$ be a common affine zero of $\mathcal{F}$ which satisfies $\mathcal{C}$. Since each of the s_2 variables in the set of s_1 homogeneous linear equations, $\hat{\mathcal{F}}$, stands for a monomial in $x_1, \cdots, x_n$, this solution for $\mathcal{F}$ generates a solution for $\hat{\mathcal{F}}$, say $v_1 = C_1$,$v_2 = C_2$,$\cdots$,$v_{s_2} = C_{s_2}$. So, if $m_1, \cdots, m_{s_2}$ are the $s_1 \times 1$ column vectors of the Dixon matrix N then,

$$C_1 m_1 + C_2 m_2 + \cdots + C_i m_i + \cdots + C_{s_2} m_{s_2} = 0.$$

Let Z be any element of $\mathcal{N}_1$. Since Z was obtained from N by deleting some column (say m_i) of N which belonged to $nvcol(\mathcal{C})$, it implies that the monomial labeling m_i cannot vanish on any solution which satisfies $\mathcal{C}$. In other words, $C_i \neq 0$. So by dividing the above equation by C_i and rearranging terms on both sides, we get the equation:

$$m_i = \frac{-C_1}{C_i} m_1 + \frac{-C_2}{C_i} m_2 + \cdots + \frac{-C_{i-1}}{C_i} m_{i-1} + \frac{-C_{i+1}}{C_i} m_{i+1} + \cdots + \frac{-C_{s_2}}{C_i} m_{s_2}$$

In other words, the column vector m_i is a linear combination of the column vectors $m_1, \cdots, m_{i-1}, m_{i+1}, \cdots, m_{s_2}$. Hence, the dimension of the vector space spanned by columns $m_1, \cdots, m_{i-1}, m_{i+1}, \cdots, m_{s_2}$ is the same as the dimension of the vector space spanned by columns $m_1, \cdots, m_{s_2}$. As a consequence, the rank of the matrix Z formed by the columns $m_1, \cdots, m_{i-1}, m_{i+1}, \cdots, m_{s_2}$ must be the same as the rank of the matrix n, formed by all the column vectors. I.e., $rank(Z) = rank(N)$. $\square$

Now we go back to the case when the coefficients are parameterized by the parameters A, in other words, the coefficients are polynomials in $\mathbb{Q}[a_1, \ldots, a_m]$. Let

$\mathcal{F} \subset \mathbb{Q}[a_1, \ldots, a_m, x_1, \cdots, x_n]$ be a set of $n+1$ polynomials and D the Dixon matrix of $\mathcal{F}$ (with entries now from $\mathbb{Q}[a_1, \ldots, a_m]$). Let $r = rank(D)$. Also, let $\mathcal{D}_1$ be the set of all $s_1 \times (s_2 - 1)$ matrices obtained from D by deleting a column which is an element of $nvcol(\mathcal{C})$ (just as $\mathcal{N}_1$ was obtained from N previously). Finally, let

$$\mathcal{R} = \{Y \mid Y \text{ is an } \ r \times r \ \text{ nonsingular submatrix of } \ D\}$$

Note that $\mathcal{R}$ is never an empty set because $rank(D) = r$ forces the existence of at least one non-singular $r \times r$ submatrix of D. Also, by definition, for all $Y \epsilon \mathcal{R}$, $det(Y) \not\equiv 0$.

The following theorem says that if there exists a column in the Dixon matrix of $\mathcal{F}$ which is in $nvcol(\mathcal{C})$ and whose removal reduces the rank of the Dixon matrix, then any rank subdeterminant of the Dixon matrix is a valid identically non-zero projection operator of $\mathcal{F}$ modulo $\mathcal{C}$. Formally, the theorem is stated as follows:

Theorem 3.1.2 *If $\exists Z \epsilon \mathcal{D}_1$ s.t. $rank(Z) < rank(D)$ then for all $Y \epsilon \mathcal{R}$, $det(Y)$ is an* identically non-zero *projection operator of $\mathcal{F}$ modulo the constraint $\mathcal{C}$.*

Proof : Let $\phi : \{a_1, \ldots, a_m\} \mapsto \bar{\mathbb{Q}}$ be a mapping which gives values to the parameters from the algebraic closure of $\mathbb{Q}$. By abuse of notation $\phi(\mathcal{F})$,$\phi(D)$, and $\phi(\mathcal{D}_1)$ are the results of substituting those values for the parameters in $\mathcal{F}$, D and $\mathcal{D}_1$ respectively and removing all zero rows and columns from the matrix D and the matrices in $\mathcal{D}_1$.

Let $\mathcal{G} = \phi(\mathcal{F})$, then it is easy to see that $N = \phi(D)$ is the Dixon matrix of $\mathcal{G}$ and $\mathcal{N}_1 = \phi(\mathcal{D}_1) - \{N\}$ is the set of all submatrices of N obtained by deleting a column which is an element of $nvcol(\mathcal{C})$ from N. So by Lemma 3.1.1, if $\mathcal{G} = \phi(\mathcal{F})$ has a common affine zero which satisfies $\mathcal{C}$ then

$$\forall Z \epsilon \mathcal{D}_1, rank(\phi(D)) = rank(\phi(Z)).$$

Also, the rank of any matrix with entries from $\mathbb{Q}[a_1, \ldots, a_m]$ cannot increase when values from $\bar{\mathbb{Q}}$ are substituted for the parameters $a_1, \ldots, a_m$, so

$$\forall Z \epsilon \mathcal{D}_1, rank(\phi(Z)) \leq rank(Z).$$

If there is a matrix $Z \epsilon \mathcal{D}_1$ which satisfies the premise of Lemma 3.1.1 then

$$rank(Z) < rank(D) = r,$$

and from the above equations we get,

$$\forall Z \epsilon \mathcal{D}_1, \quad rank(\phi(D)) = rank(\phi(Z)) \leq rank(Z) < rank(D) = r$$

$$\text{i.e., } rank(\phi(D)) < r.$$

This means that once the values of parameters are substituted in D, its rank becomes less than r, i.e., all the $r \times r$ submatrices of D (which are exactly the elements of $\mathcal{R}$) become singular, which means that their determinants vanish at these values of the parameters. $\square$

Theorem 3.1.2 suggests that if $\exists Z \epsilon \mathcal{D}_1$ such that $rank(Z) < rank(D)$, then a identically non-zero projection operator, $\varrho_{\mathcal{C}}$, is the determinant of any matrix in $\mathcal{R}$. But we need to be able to efficiently perform the check, obtain an element of $\mathcal{R}$ and finally compute its determinant. Next we prove a lemma which will give an efficient method to perform the check.

Let $\bar{w} = (w_1, \cdots, w_{s_2})^T$ (where w_i are elements from the same domain as the entries of D) be the $s_2 \times 1$ vector which satisfies the matrix equation $D\bar{w} = \bar{0}$, then

Lemma 3.1.3 *$\exists Z \epsilon \mathcal{D}_1$ s.t. $rank(Z) < rank(D)$ if and only if there exists some $0 \leq i \leq s_2$ such that $w_i = 0$ and the monomial labeling the i^{th} column of the Dixon matrix cannot vanish at any point satisfying the constraint $\mathcal{C}$.*

Proof : First of all, let us write the expanded version of the matrix equation $D\bar{w} = \bar{0}$:

$$m_1 w_1 + m_2 w_2 + \cdots + m_{s_2} w_{s_2} = \bar{0}$$

Only if: Let $Z \epsilon \mathcal{D}_1$ *s.t.* $rank(Z) < rank(D)$. Let m_i be that row of D whose deletion resulted in Z. Then, by definition of $\mathcal{D}_1$, m_i cannot vanish at any point satisfying $\mathcal{C}$. Also, $rank(Z) < rank(D)$, implies that m_i is linearly independent. This means that $w_i = 0$ because otherwise, the above equation can be divided by w_i and then by rearranging terms, m_i can be expressed as a linear combination of other columns, hence implying that m_i is linearly dependent and resulting in a contradiction.

If: Assume that there exists an $0 \leq i \leq s_2$ for which $w_i = 0$ and m_i cannot vanish at any point satisfying $\mathcal{C}$, i.e., $m_i \in nvcol(\mathcal{C})$. The latter assumption means that the matrix obtained by deleting m_i from D (say Z) must be in $\mathcal{D}_1$. But also since $w_i = 0$, it follows that column m_i is linearly independent of other columns of D, hence implying that $rank(Z) < rank(D)$. □

Finally, in accordance with Theorem 3.1.2, we would like to compute the determinant of any element in $\mathcal{R}$ which will give the required projection operator in the parameters. We now prove a lemma which will give an efficient method to compute this determinant. Let D_{row} have the following properties:

1. D_{row} is row-reduced, i.e., each column of D_{row} which contains the leading non-zero entry of some row has all its other entries 0.

2. D_{row} is row-equivalent to D, i.e., D_{row} can be obtained from D by a finite sequence of the following two steps:

   (a) **Elimination step:** Replacement of i^{th} row of D by the i^{th} row plus d times j^{th} row, where d is any rational function in the parameters, and $i \neq j$.

   (b) **Pivoting step:** Interchange two rows (say the i^{th} and the j^{th} rows) of D.

Notice that D_{row} can be constructed from D by simple Gaussian elimination and many computer algebra systems already support such an operation. In MAPLE, D_{row} can be easily computed using the procedure $gausselim(D)$.

Lemma 3.1.4 *There exists some $Y \epsilon \mathcal{R}$ such that the product of all the pivot elements of D_{row} is equal to $det(Y)$.*

Proof : Assign to the j^{th} row of D, a label l_j. Now obtain the matrix D_{row} which is row-equivalent to D by successively applying a permutation of the above two steps, except that whenever the pivoting step is applied, interchange the labels of the rows too. In the elimination step, the labels remain the same. Since the rank of D is r, D_{row} will have r pivots. Let $m_{i_1}, \cdots, m_{i_r}$ be the pivot columns, and $l_{j_1}, \cdots, l_{j_r}$ the labels of pivot rows of D_{row}. Then it is easy to show that the product of all pivot elements of D_{row} is the determinant of the $r \times r$ submatrix (say Y) of D obtained by the rows labeled by $l_{j_1}, \cdots, l_{j_r}$ and the columns indexed by $m_{i_1}, \cdots, m_{i_r}$ in D. Since the product of pivot elements is identically non-zero, Y is not singular so it is in $\mathcal{R}$ by definition. $\square$

Assume that the set of $n+1$ polynomials $\mathcal{F} \subset \mathbb{Q}[a_1, \ldots, a_m, x_1, \cdots, x_n]$ and the constraints $\mathcal{C}$ are given. The following algorithm, which we call the *RSC* (Rank Submatrix Computation), based on Theorem 3.1.2 and Lemmas 3.1.3 and 3.1.4, checks if the precondition in Theorem 3.1.2 is true in which case it returns $\varrho_{\mathcal{C}}$. If the precondition of Theorem 3.1.2 is not true, then our technique may fail to extract a non-zero projection operator.

1. Set up the $s_1 \times s_2$ Dixon matrix (D) of $\mathcal{F}$ using the steps of the classical Dixon formulation.

2. Solve the matrix equation $D\bar{w} = \bar{0}$. Let $\bar{w} = (w_1, \ldots, w_{s_2})$ denote the solution.

3. Find out if there exists a w_i in $\bar{w}$ such that $w_i = 0$ and also that the monomial labeling m_i cannot vanish at any point satisfying $\mathcal{C}$. If such a w_i exists then

(a) Compute D_{row}.

(b) Return the product of all the pivots of D_{row}.

4. Else, this heuristic fails.

Theorem 3.1.5 *If the Dixon matrix of a set of polynomials satisfies the precondition of Theorem 3.1.2, then the* RSC *algorithm computes $\varrho_{\mathcal{C}}$, a multiple of the resultant of $\mathcal{F}$ which is identically non-zero, and which vanishes on all those particular values of parameters from $\bar{\mathbb{Q}}^m$ for which $\mathcal{F}$ has a common affine zero in $\bar{\mathbb{Q}}^n$ and that common zero also* satisfies the set of constraints $\mathcal{C}$.

Proof : Follows directly from Theorem 3.1.2 and Lemmas 3.1.3 and 3.1.4. □

There are two points to be noted here. First, though there exist examples where the condition in step (3) is not true, they are rare. In all the examples that we tried in this thesis, D was singular or rectangular many times, but the condition of step (3) always held. Secondly, once the polynomial in the parameters is obtained from the *RSC* algorithm, it can usually be factored. Since the vanishing of this polynomial is only a necessary condition, occasionally it will be the case that some of the factors do not vanish on any of the solutions. These factors can be removed to obtain a smaller $\varrho_{\mathcal{C}}$. This is done manually as suggested by the specific problem. Our experience has been that it is usually easy to identify such extraneous factors, but an automatic method to accomplish this needs to be further explored.

Example: (Dixon/*RSC*): Recall that in our example illustrating the classical Dixon formulation in the previous chapter, we had ended up with a 2×3 Dixon matrix. With the algorithm outlined above, we can extract a non-zero projection operator from this rectangular Dixon matrix as follows.

It is easily confirmed that the first column of the rectangular Dixon matrix is linearly independent of the other columns, hence the rank of the matrix (2) decreases (becomes 1) on its removal. Perform Gaussian elimination on the Dixon matrix, and the projection operator is the product of the 2 pivots. It has 75 terms (in expanded form):

$$\varrho = -b\rho\,(a, b, c)\,.$$

There is one extraneous factor but among all four methods for this example, the reader might observe that this method generates the least number of extraneous factors. $\square$

3.2 *RSC:* A Sparsity Preserving Alternative to Perturbation

Before we discuss the applicability of *RSC* to other resultant formulations, we outline another approach given by Canny [5] for dealing with degenerate matrices in the Macaulay formulation.

3.2.1 Generalized Characteristic Polynomials (*GCP*)

One way to deal with degenerate resultant matrices is to perturb the input polynomials so that the resultant matrix of the perturbed polynomial system is square and nonsingular [22, 5, 9], and then extract a non-trivial multiple of the resultant of the original system from it. One can try ad hoc perturbations of $\mathcal{F}$ in the hope of obtaining a nontrivial projection operator as follows. Randomly perturb the system using a perturbation variable, compute the projection operator for this perturbed system, factor it, remove any extraneous factors and finally, substitute zero for the perturbation variable to get the projection operator of the original system. Such ad hoc perturbations can be used in conjunction with all three resultant formulations, but they may not work. The following general perturbation mechanism applicable to Macaulay formulation was given by Canny in [5].

Let s be a new perturbation variable. Create a new set of polynomials $\mathcal{F}(s)$ by replacing each $p_j \in \mathcal{F}$ (for $1 \leq j \leq n$) by $p_j - s x_j^{d_j}$ and p_{n+1} by $p_{n+1} - s$. Compute the projection operator, $\varrho(s)$, of $\mathcal{F}(s)$ using Macaulay formulation. $\varrho(s)$ is known as the *Generalized Characteristic Polynomial* (*GCP*) and is a polynomial in s. The trailing coefficient (w.r.t. s) of *GCP* is a nontrivial projection operator of $\mathcal{F}$. Note that the size of the Macaulay matrix is same for $\mathcal{F}$ and $\mathcal{F}(s)$. For a proof of the fact that the projection operator so derived is identically non-zero and it does vanish on all the affine zeros of the system, see [5]. We call this method **Macaulay/**GCP. We now finish the previous example using this method.

Example: (Macaulay/GCP**)** In the example from the previous chapter which was used to illustrate the Macaulay formulation, both $det(M_\sigma)$ and $det(N_\sigma)$ were identically zero. So we compute the *GCP*. Construct $\mathcal{F}(s)$ and letting $M_\sigma(s)$ and $N_\sigma(s)$ stand for its Macaulay and the denominator matrix,

$$
\begin{aligned}
det(N_\sigma(s)) &= det(N_\sigma - Is) = s(s-a), \\
det(M_\sigma(s)) &= det(M_\sigma - Is).
\end{aligned}
$$

$det(M_\sigma(s))$ is a polynomial with 705 terms. $GCP = det(M_\sigma(s))/det(N_\sigma(s))$ is a polynomial with 462 terms. Finally, the projection operator, which here is the constant term with respect to s of the GCP, has 159 terms (in expanded form):

$$
\varrho = c\,(a+b)\left(a^2c + abc - 4ab + 4b^2\right)\rho\,(a,b,c)
$$

Notice that there are 3 extraneous factors. □

3.2.2 Disadvantage of Perturbation Methods

As was stated earlier, the GCP method can probably be modified to work in conjunction with other formulations such as Dixon and sparse. However, this method can possibly lead to inefficiency in resultant computation. As we have alluded to earlier, sparse resultant formulation was developed to exploit the structure of the Newton polytopes of the input polynomials to construct smaller resultant matrices. We will show later in this thesis that even the Dixon formulation exploits the polytope structure to a certain extent, for efficient resultant computation. So, any method which works by adding monomials to the input polynomials, risks increasing the volumes of the input polytopes, and possibly making the resultant computation process costlier.

The *GCP* method works by adding terms to polynomials thereby changing their structure. Our experience has shown that this indeed destroys the sparsity in the input system by increasing the volume of their polytopes. This fact may not be significant in the Macaulay formulation which does not exploit the sparsity in the polynomials, but for Sparse resultant formulation and the Dixon formulation, whose complexity depends on the volumes of the input polytopes, such perturbations can lead to inefficiency.

There is another way in which perturbation based methods such as *GCP* can adversely affect the efficiency, even for the Macaulay formulation. Perturbation is simulated by introducing a new parameter (s in the above example). This increases the number of parameters, thereby making it more difficult to extract the resultant from the resultant matrix. In fact, in the *GCP* method, the terms (with s as the coefficient) are added at the highest degree in each input polynomial. This ensures that the resultant *after* perturbation has a degree in the perturbation variable s as large as the size of the Macaulay matrix. This increases the size of the resultant of the perturbed system by a substantial amount, making it difficult to compute it.

In the chapter containing empirical results, we will show that, in fact, such perturbations almost render the *GCP* method useless.

3.2.3 Applicability of *RSC* to Other Formulations

The *RSC* method which we outlined in the previous section, is actually applicable to all formulations. Let us assume that we are interested only in the existence of solutions in the algebraic torus, then the *RSC* algorithm becomes the following:

1. Set up the Resultant matrix (R) of $\mathcal{F}$.

2. If any column in R is linearly independent of the other columns, then return the determinant of any *rank submatrix* of R.

3. Else, this heuristic fails.

The resultant matrix R in the above algorithm can be a resultant matrix from any of the three formulations, giving three methods for computing a nontrivial projection operator, viz. **Macaulay/***RSC*, **Dixon/***RSC* and **Sparse/***RSC*. We have already illustrated this method on the Dixon formulation in the previous section, and now we illustrate it on the Macaulay formulation.

Example: (Macaulay/*RSC***)** It is easily confirmed that the last column of the singular Macaulay matrix, M_σ, from the previous chapter is linearly independent of the other columns, hence the rank of the matrix (9) decreases (becomes 8) on its removal. Projection operator is the product of the 9 pivots after Gaussian elimination and has 159 terms (in expanded form):

$$\varrho = ac\,(a+b)\left(a^2c + abc - 4ab + 4b^2\right)\rho\,(a,b,c)\,.$$

Notice that there are 4 extraneous factors in contrast to Macaulay/*GCP*, which had 3 extraneous factors. □

The major advantage of our technique is in the fact that it does not alter the structure of the input polynomial system, thus preserving the sparsity in it. Consequently, the size of the resultant matrix, or the complexity of the problem does not change at all. *RSC* can be viewed as the back-end of any resultant formulation: Set up the resultant matrix using the formulation, then if the resultant matrix is degenerate, use *RSC* to extract a non-trivial projection operator. *RSC* extracts a non-trivial projection operator while leaving the polynomial system intact, thus enabling the underlying resultant formulation to exploit the sparsity.

Another issue is that of extraneous factors. As is evident from the examples in this and previous chapters, all methods we discussed lead to extraneous factors. In fact, for the example that we have been using, the *RSC* method, as applied to the Dixon formulation generated the least number of extraneous factors. Nevertheless, we are not in a position to derive any bounds on the number of extraneous factors in the various methods, much less eliminate the extraneous factors from any of the discussed methods. So, while it is desirable to minimize extraneous factors, due to a lack of results on extraneous factors, they cannot be a consideration in making a choice among the various state of the art methods for computing a non-zero projection operator.

3.3 Some Crude Bounds

The size of the resultant matrix is critical in determining the efficiency of a given method. Smaller resultant matrices lead to efficient computation of their determinant or rank subdeterminant. In this section, we use very simple analysis to compute some bounds on the sizes of the various resultant matrices. We compare the sizes of the three resultant matrices in the worst case using two measures:

n	d	Dixon	Sparse	Macaulay
2	2	5	15	15
2	3	12	33	36
2	4	22	66	66
3	2	14	54	56
3	3	55	198	220
4	2	42	179	210

Table 3.1: Matrix Sizes, Worst Total Degree Case.

1. the total degree of the input polynomials and
2. the degree of input polynomials in each variable.

3.3.1 Total Degree based

For $1 \leq i \leq (n+1)$, let $tdeg(p_i, X) = d$. It can be easily established that the sizes of Dixon, Macaulay and sparse resultant matrix are bounded by $\binom{nd}{n}$, $\binom{(n+1)d}{n}$ and $\binom{(n+1)d}{n}$ respectively.

It should again be pointed out that "sparse" resultant formulation is really a misnomer because the sparsity is in this formulation is measured by the volumes of the input polytopes – not the number of terms with non-zero coefficients in the input polynomials. We illustrate this point further as follows:

As is evident from these bounds, if a polynomial systems is dense, that is, if all terms of degree less or equal to that of the polynomials have a non-zero coefficient, then the sparse matrix becomes identical to the Macaulay matrix. However it is not necessarily the case that if some or most of these terms were missing from the polynomials (that is they have zero coefficients), then the size of their sparse resultant matrix would be much smaller than the worst case size of dense polynomials (that is the size of their Macaulay matrix). As long as the volume of the polynomial system is the same as that of the polynomials in the worst case, the sparse matrix size remains the same as the Macaulay matrix size. Therefore, even though the algorithm for sparse resultant is not completely insensitive to zero terms (especially the greedy and incremental versions), the worst case matrix size is easily achieved for polynomials with relatively small number of terms.

For example, for $n = 2, d = 3$, a set of three polynomials with the supports $(x^3, x^2, xy^2, y^3, 1)$ are relatively sparse (only 5 terms as opposed to maximum possible 10), but the sparse matrix still has the worst case size, as if the polynomials were completely dense of total degree 3.

More efficient algorithms for sparse resultants use heuristics which may construct matrices of smaller size than the formula above indicates. However, such heuristics are not considered here because the order of the bound does not change. The formula for Dixon matrix size given above is loose and the exact size is always smaller.

These formulas show that the Dixon matrix is smaller than the other two. See Table 3.1 for typical worst case sizes in practice.

The total degree measure is not very fair to both Dixon and sparse resultants. Better bounds are obtained for these two if the measure for the degree of each polynomial in individual variables is considered.

n	d	Dixon	Sparse	Macaulay
2	2	8	32	66
2	3	18	72	153
2	4	32	144	276
3	2	48	488	2024
3	3	162	1263	7140
4	2	384	*	91390

Table 3.2: Matrix Sizes, Worst Ind. Var. Deg. Case.

3.3.2 Variable Degree based

For $1 \leq i \leq (n+1)$ and $1 \leq j \leq n$, let $d = deg(p_i, x_j)$. Sizes of the Dixon, Macaulay and sparse resultant matrices are bounded by $n!d^n$, $\binom{(n+1)nd}{n}$ and $((n+1)d)^n$ respectively.

Using these formulas and Stirlings approximation for factorial (for asymptotically large n and d), it can be proven that Dixon matrix is smaller, by an exponential $(\mathbf{O}\left(e^{n+1}\right))$ factor, than the sparse resultant matrix, which in turn is smaller than the Macaulay matrix also by an exponential $(\mathbf{O}\left(e^{n}\right))$ factor. The Dixon matrix is thus smaller than the Macaulay matrix by an exponential $(\mathbf{O}\left(e^{2n+1}\right))$ factor. Typical worst case sizes are shown in Table 3.2. The (*) entry means the program to construct the sparse resultant matrix ran out of memory for the case of $n = 4, d = 2$.

3.4 Summary

In this chapter, we extended the applicability of Dixon's classical formulation for computing resultants. We showed how RSC can be used in the case of degenerate resultant matrices occurring in other formulations also. We argued that RSC offers a much more efficient alternative to perturbation methods, which have traditionally been used with formulations such as Macaulay. Finally, we derived some loose bounds on the sizes of the Dixon matrix and compared them to similar bounds on other resultant matrices. These bounds were derived for two different cases, but both cases were examples of completely dense systems.

However, polynomial systems which arise in the real world are often very sparse. For any resultant method to be really effective, it must exploit this sparsity. The loose bounds which we have derived till now only consider the case of dense polynomial systems and do not show if the Dixon formulation and our extension of it exploits sparsity at all. In fact, since the Dixon formulation is classical, developed early this century, whereas the sparsity analysis such as development of BKK bounds [3] on the number of affine roots of polynomial systems is fairly recent (1970s), one may be led to conclude that the Dixon method is immune to the existence of sparsity in a problem, i.e., the sizes of the Dixon matrices *do not* depend on the sparsity in the input polynomial system. As we will show in the next chapter, this is far from the truth. In the next two chapters, we establish a definite connection between the sparsity of input polynomials as determined by their Newton polytopes, and the complexity of Dixon's formulation.

Chapter 4

Sparsity Considerations

Linking Dixon Formulation to Input Polytopes

Classical elimination theory traditionally deals with homogeneous polynomials, which can be characterized merely by their degree. Classical Bezout theorem says that the number of common projective solutions (in the algebraic closure of the coefficient field) of n homogeneous polynomials in n variables is either infinite or the product of their degrees, provided the multiplicity of each solution is counted properly. In fact, Macaulay had developed his formulation of the resultant as an effort to characterize the conditions for the existence of non-trivial solutions to homogeneous polynomials. That is why the projection operator computed by Macaulay is known as the *homogeneous resultant*. The vanishing of the homogeneous resultant of n homogeneous polynomials in n variables is a necessary and sufficient condition for them to have a common non-trivial solution (or a projective zero). Since homogeneous polynomials can only be characterized by their degree, the size of the Macaulay resultant matrix, and the complexity of Macaulay's formulation in general, is dependent on the total degrees of the input polynomials, or their Bezout bound.

Since the beginning of this thesis, we have been interested in non-homogeneous polynomials. When dealing with non-homogeneous polynomials, the traditional approach is to homogenize them using a new *homogenizing variable*, and then apply the classical elimination theory for homogeneous polynomials. However, if one is interested in the affine solutions of a non-homogeneous set of polynomials, the Bezout bound only serves as an upper bound, and is generally quite loose. Likewise, the Macaulay formulation yields a resultant whose vanishing is a condition for existence of *projective roots*, not necessarily affine. This leads to two shortcomings in using the classical elimination theory for non-homogeneous systems: (1) The homogeneous resultant may vanish even when there are no affine solutions and (2) the complexity of resultant computation depends on the Bezout bound, which may be too loose for non-homogeneous systems, leading to inefficiency. The *GCP* method developed by Canny, and our *RSC* method described in the last chapter provide means to deal with problem (1). This chapter concerns the issues raised by problem (2).

Sparse elimination theory was developed in mid 1970s to prove stronger and tighter results for non-homogeneous polynomial systems by considering their structure as described by their Newton polytopes, not their total degrees. Newton polytopes precisely represent the combinatorial structure of non-homogeneous polynomials. The major breakthrough in sparse elimination theory was the discovery of a link between the Newton polytopes of non-homogeneous polynomials and their roots in the *algebraic torus*, or $\mathbb{C}^* = (\mathbb{C} - 0)$. Results by Bernstein, Kushnirenko and Khovanskii [3] linked the number of affine common solutions in ${\mathbb{C}^*}^n$ of a set of n non-homogeneous polynomials in n variables to their Newton polytopes, thus providing a much tighter bound than the Bezout bound. Likewise, the sparse resultant was defined by Gelfand et. al. [23] and Sturmfels [40] to generalize the concept of resultant for the existence of roots of non-homogeneous polynomials in ${\mathbb{C}^*}^n$. Algorithms to compute the sparse resultant were developed by Emiris [14]. The advantage of these algorithms was not only that they provided conditions on the existence of affine solutions of non-homogeneous polynomials, but also that their complexity depended on the structure of the polynomials, as determined by their Newton polytopes – not total degree. The size of the sparse resultant matrix can be determined by computing certain volumes of the Newton polytopes of the input polynomials. This resulted in considerable gains in efficiency, as demonstrated by Emiris [14]. Sparse resultant effort was thus the first attempt at *explicitly* exploiting the sparsity in input non-homogeneous polynomials, as determined by their Newton polytopes.

Since the classical Dixon formulation was developed early this century during the era of classical elimination theory, it was viewed as another alternative, though efficient, method to compute the homogeneous resultant. It was analyzed only for the absolutely dense cases, such as the class of generic n-degree systems, because those were the only cases for which this method was guaranteed to work. Naturally, analysis of such dense cases did not reveal any dependence of the Dixon method on the sparsity of input polynomials. Therefore it was unknown whether the Dixon formulation exploits the sparse structure of the polynomials as determined by their Newton polytopes. However, our extension of the Dixon formulation to polynomial systems which are not generic or n-degree, in other words, possibly sparse, makes it imperative to determine if our extension of this formulation at all exploits the sparse structure of the input polynomials. In this chapter, we will show that this is indeed the case. We prove that the size of the Dixon matrix is actually determined by the structure of the Newton polytopes of the input polynomials, not their total degree. In fact, by exploiting this relationship, we prove that the Dixon matrix is smaller than even the sparse resultant matrix. In the next chapter, we show that the complexity of computing the resultant using the extended Dixon formulation also depends on Newton polytopes of the input polynomials, and is lower than that of using sparse or the Macaulay formulation. Our results show that the extended Dixon formulation *implicitly* exploits the sparsity in the input non-homogeneous polynomials, as determined by their Newton polytopes.

4.1 Newton Polytope of the Dixon Polynomial

Recall that in the Dixon formulation, we start with a set $\mathcal{F} = \{p_1, \ldots, p_{n+1}\}$ of $n+1$ polynomials in n variables $X = \{x_1, \ldots, x_n\}$. For simplicity, we will assume that the coefficient field of the polynomials is that of rational numbers. Assuming it to be the field of rational functions in the parameters does not change the analysis of this section at all. The objective is to eliminate the variables X from $\mathcal{F}$. Recall that another set $\bar{X} = \{\bar{x}_1, \ldots, \bar{x}_n\}$ of variables disjoint from X is used for substitution in the cancellation matrix. Given a polynomial $p \in \mathbb{Q}[X]$, $p(\bar{x}_1, \ldots, \bar{x}_k, x_{k+1}, \ldots, x_n)$, which appear as entries of the cancellation matrix, stand for uniformly replacing x_j by $\bar{x}_j$ for $1 \leq j \leq k$ in p. For a given polynomial $q \in \mathbb{Q}[X, \bar{X}]$, $V_X(q)$ (or $V_{\bar{X}}(q)$), the **monomial vector** of q w.r.t. X (or $\bar{X}$), is defined to be the vector of all monomials appearing in q, when viewed as a polynomial in the variables X (or $\bar{X}$). The **exponent vector** of a monomial $x_1^{e_1} \cdots x_n^{e_n} \bar{x}_1^{\bar{e}_1} \cdots \bar{x}_n^{\bar{e}_n}$ w.r.t. the set X of variables is defined to be the n-tuple $(e_1, \ldots, e_n) \in \mathbb{Z}^n$. The **support**, $supp(q) \subset \mathbb{Z}^n$, of a polynomial q is the set of exponent vectors of all the monomials, w.r.t. X, with non-zero coefficients in q.

Example: If $X = \{x_1, x_2\}$ and

$$p = 2x_1^3 x_2 \bar{x}_1^3 + x_2^4 \bar{x}_1 \bar{x}_2^2 + \bar{x}_1^2 + \bar{x}_1 \bar{x}_2 + 1 \in \mathbb{Q}[X, \bar{X}],$$

then

$$\begin{aligned} V_X(p) &= \left[x_1^3 x_2,\ x_2^4,\ 1\right], \\ V_{\bar{X}}(p) &= \left[\bar{x}_1^3,\ \bar{x}_1 \bar{x}_2^2,\ \bar{x}_1^2,\ \bar{x}_1 \bar{x}_2,\ 1\right] \quad \text{and} \end{aligned}$$

$$supp(p) \quad = \quad \{(3,1),(0,4),(0,0)\}.$$

Notice that support is always with respect to X. $\square$

Let the convex hull of a set $\mathcal{A} \subset \mathbb{Z}^n$ be denoted by $conv(\mathcal{A}) \subset \mathbb{R}^n$. The **Newton polytope** of a polynomial $p \in \mathbb{Q}[X, \bar{X}]$ is defined to be $\mathcal{N}(p) = conv(supp(p)) \subset \mathbb{R}^n$. It is very important to note that we have defined the Newton polytope **only with respect to the variable set** X, while variables in $\bar{X}$ are treated as constants – just as was the case while defining $supp()$. That is why $\mathcal{N}(p) \subset \mathbb{R}^n$. For any polynomial p, it is obvious that

$$supp(p) \subseteq \mathcal{N}(p) \cap \mathbb{Z}^n.$$

Given a set of points $\mathcal{A} \subset \mathbb{R}^n$ and an integer $0 \leq i \leq n$, $\pi_i(\mathcal{A}) \subset \mathbb{R}^n$ stands for the set obtained by replacing the first i coordinates of all points in $\mathcal{A}$ by 0. Geometrically, $\pi_i(\mathcal{A})$ represents a projection of the set $\mathcal{A}$ to the *last* $n-i$ dimensions by *flattening* the first i dimensions. We will abuse the terminology and call this projection as the $\mathbf{i^{th}}$ **projection** of $\mathcal{A}$.

Example: For any $\mathcal{A} \subset \mathbb{R}^n$, $\pi_0(\mathcal{A}) = \mathcal{A}$ and $\pi_n(\mathcal{A}) = \{0, \ldots, 0\}$. $\square$

Example: For the polynomial p in the previous example,

$$\pi_1\left(\mathcal{N}\left(p\right)\right) = \mathcal{N}\left(\{(0,4),(0,0)\}\right).$$

$\square$

The **Minkowski sum** of two sets $\mathcal{A}, \mathcal{B} \in \mathbb{R}^n$ is $\mathcal{A} + \mathcal{B} = \{a + b \mid a \in \mathcal{A}, b \in \mathcal{B}\} \subset \mathbb{R}^n$, or equivalently,

$$\mathcal{A} + \mathcal{B} = \bigcup_{a \in \mathcal{A}} (a + \mathcal{B}). \tag{4.1}$$

Example: For the polynomial p in the previous example,

$$\mathcal{N}(p) + \pi_1(\mathcal{N}(p)) = \mathcal{N}(\{(3,5),(3,1),(0,0),(0,8)\}).$$

□

The following two facts about Newton polytopes can be easily proved:

Fact 4.1.1 *Given a set $\mathcal{A} \subset \mathbb{R}^n$ and two polynomials p and q, if $\mathcal{N}(p) \subseteq \mathcal{A}$ and $\mathcal{N}(q) \subseteq \mathcal{A}$, then $\mathcal{N}(p+q) \subseteq \mathcal{A}$.*

Fact 4.1.2 *Given two polynomials p and q,*

$$\mathcal{N}(p\,q) = \mathcal{N}(p) + \mathcal{N}(q).$$

Recall that the **cancellation matrix** of $\mathcal{F}$, denoted Δ, is

$$\Delta = \begin{bmatrix} p_1(x_1, x_2, x_3, \ldots, x_n) & \cdots & p_{n+1}(x_1, x_2, x_3, \ldots, x_n) \\ p_1(\bar{x}_1, x_2, x_3, \ldots, x_n) & \cdots & p_{n+1}(\bar{x}_1, x_2, x_3, \ldots, x_n) \\ p_1(\bar{x}_1, \bar{x}_2, x_3, \ldots, x_n) & \cdots & p_{n+1}(\bar{x}_1, \bar{x}_2, x_3, \ldots, x_n) \\ \vdots & \vdots & \vdots \\ p_1(\bar{x}_1, \bar{x}_2, \bar{x}_3, \ldots, \bar{x}_n) & \cdots & p_{n+1}(\bar{x}_1, \bar{x}_2, \bar{x}_3, \ldots, \bar{x}_n) \end{bmatrix}.$$

The **Dixon polynomial** of $\mathcal{F}$, denoted $\delta \in \mathbb{Q}[X, \bar{X}]$, is

$$\delta = \frac{|\Delta|}{\prod_{i=1}^{n}(x_i - \bar{x}_i)},$$

and the **Dixon matrix** of $\mathcal{F}$ is a matrix D for which

$$\delta = V_{\bar{X}}(\delta)\, D\, V_X(\delta)^T.$$

As is evident from the above expression, the Dixon matrix, intuitively, is the collection of all coefficients of the Dixon polynomial. Since each term in the Dixon polynomial is basically a product of a monomial in X and another in $\bar{X}$, V_X and $V_{\bar{X}}$ basically represent the collection of monomials in X and $\bar{X}$ in the Dixon polynomial, whose products form the terms of the Dixon polynomial. The coefficient of the term corresponding to the product of the i^{th} element of $V_{\bar{X}}$ and the j^{th} element of V_X in the Dixon polynomial is the $(i, j)^{th}$ entry of the Dixon matrix. So if this term does not exist in the Dixon polynomial, $D[i, j] = 0$.

In the following discussion, we assume that the input system is *unmixed*, which is defined as follows.

Definition 4.1.3 *A set of polynomials is said to be* **unmixed** *if all its polynomials have the same Newton polytope.*

We would like to point out that unmixed systems form an important subclass which closely approximate polynomial systems occurring in various application domains. This will be demonstrated in later chapters as we approximate the polynomial systems occurring in many applications by unmixed systems to derive tight bounds on their resultant matrix sizes.

For an unmixed polynomial set $\mathcal{F}$, let $\mathcal{N}(\mathcal{F})$ denote the Newton polytope of any polynomial in $\mathcal{F}$. The following lemma relates the Newton polytope of the determinant of the cancellation matrix to the Newton polytopes of input polynomials.

Lemma 4.1.4 *For an unmixed set $\mathcal{F} \subset \mathbb{Q}[X]$ of $n+1$ polynomials,*

$$\mathcal{N}\left(|\Delta|\right) \subseteq \sum_{i=0}^{n} \pi_i\left(\mathcal{N}(\mathcal{F})\right),$$

where Δ is the cancellation matrix of $\mathcal{F}$.

Proof : For any given polynomial $q \in \mathbb{Q}[x_1, \ldots, x_n]$,

$$\mathcal{N}\left(q\left(\bar{x}_1, \ldots, \bar{x}_i, x_{i+1}, \ldots, x_n\right)\right) = \pi_i\left(\mathcal{N}\left(q\right)\right).$$

Therefore, since $\mathcal{F}$ is assumed to be unmixed, the Newton polytope of each entry in row $i+1$ of the cancellation matrix is $\pi_i\left(\mathcal{N}\left(\mathcal{F}\right)\right)$.

Now, in the direct expansion of the determinant $|\Delta|$ as a sum of products, each summand is a product of exactly $n+1$ polynomials, one from each row of the cancellation matrix. By Fact 4.1.2 the Newton polytope of the product of polynomials is the Minkowski sum of their individual Newton polytopes. Therefore, the Newton polytope of each summand is $\sum_{i=0}^{n} \pi_i\left(\mathcal{N}(\mathcal{F})\right)$. By Fact 4.1.1, since the Newton polytope of each summand in the expansion of $|\Delta|$ is contained in $\sum_{i=0}^{n} \pi_i\left(\mathcal{N}(\mathcal{F})\right)$, so is the Newton polytope of $|\Delta|$. □

The above lemma relates the structure of the determinant of the Cancellation matrix to the input Newton polytopes. The Dixon polynomial is obtained by dividing this determinant by $\prod_{i=1}^{n}(x_i - \bar{x}_i)$. We will use this relationship to relate the structure of the Dixon polynomial and hence the size of the Dixon matrix to the input Newton polytopes. In particular, in the following theorem, we prove that the Newton polytope and the support of the Dixon polynomial lies *inside* the polytope obtained after *perturbing* the Newton polytope of the determinant of the Cancellation matrix ($|\Delta|$) slightly towards the negative octant. This relationship bounds the size of the Dixon matrix by the number of integer lattice points inside the polytope obtained after perturbing the Newton polytope of $|\Delta|$ just a little bit (a concept formalized below) in the negative octant. Formally, the theorem states:

Theorem 4.1.5 *For an unmixed set $\mathcal{F} \subset \mathbb{Q}[X]$, of $n+1$ polynomials,*

$$\begin{aligned} \mathit{supp}(\delta) &\subseteq \left(\sum_{i=0}^{n} \pi_i\left(\mathcal{N}(\mathcal{F})\right) - \epsilon\right) \cap \mathbb{Z}^n, \quad \textit{i.e.,} \\ \textit{Number of columns in the Dixon matrix} &\leq \left|\left(\sum_{i=0}^{n} \pi_i\left(\mathcal{N}(\mathcal{F})\right) - \epsilon\right) \cap \mathbb{Z}^n\right|, \end{aligned}$$

where $\epsilon = \langle a_1, \ldots, a_n \rangle$ is any vector such that $\sum_{i=1}^{n} a_i \leq 1$ and $a_i \geq 0$ where $1 \leq i \leq n$.

Proof : Let $q = \prod_{i=1}^{n}(x_i - \bar{x}_i)$. Clearly, the coefficient the term $\bar{x}_1 \cdots \bar{x}_n$ in the polynomial q is not 0. Since $\bar{x}_1 \cdots \bar{x}_n$ is a constant w.r.t. the set X of variables, $(0, 0, \ldots, 0) \in \mathcal{N}(q)$. In fact, the origin would lie even inside the polytope obtained by perturbing the Newton polytope of q slightly in the negative direction. I.e., if ϵ is a sufficiently small vector in the positive octant, as defined in the statement of this theorem, then $(0,0,\ldots,0) \in \mathcal{N}(q) - \epsilon$. Now, $|\Delta| = q \times \delta$, and therefore,

$$\begin{aligned}
\sum_{i=0}^{n} \pi_i\left(\mathcal{N}(\mathcal{F})\right) - \epsilon &\supseteq \mathcal{N}\left(|\Delta|\right) - \epsilon = \mathcal{N}\left(q \times \delta\right) - \epsilon \quad \text{(by Lemma 4.1.4)} \\
&= \mathcal{N}(q) + \mathcal{N}(\delta) - \epsilon \qquad \text{(by Fact 4.1.2)} \\
&= (\mathcal{N}(q) - \epsilon) + \mathcal{N}(\delta) \\
&= ((\mathcal{N}(q) - \epsilon) \cup \{(0,0,\ldots,0)\}) + \mathcal{N}(\delta) \\
&= (\mathcal{N}(q) + \mathcal{N}(\delta) - \epsilon) \cup \mathcal{N}(\delta) \quad \text{(by Eqn 4.1)} \\
&\supseteq \mathcal{N}(\delta) \supseteq supp(\delta).
\end{aligned}$$

The number of columns in the Dixon matrix equals the number of monomials with non-zero coefficients in the Dixon polynomial when viewed as a polynomial in x_is, which equals the cardinality of the support of the Dixon polynomial, and the result follows. □

Intuitively, this small perturbation by ϵ is the result of the process of division by $q = \prod_{i=1}^{n}(x_i - \bar{x}_i)$. It is this division which perturbs the whole Newton polytope of the determinant of the Cancellation matrix by a very small amount towards the negative octant. The reason is that since all monomials in X which have degree at most 1 in any variable have a non-zero coefficient in q, division by q will remove all terms in which any variable is at its highest degree in the divisor. This corresponds to a slight perturbation of $\sum_{i=0}^{n} \pi_i\left(\mathcal{N}(\mathcal{F})\right)$ in the negative direction along each axis. The easiest way to visualize this is in 1 dimension, where a division with $x_1 - \bar{x}_1$ will always remove the leading term, thus perturbing the divisor in the negative direction.

This perturbation does have a definite effect on the bounds for the size of the Dixon matrix. Clearly, this slight perturbation removes some of the integral points (which lay on the boundary of the Newton polytope) outside the polytope, *without* introducing any new integral points. Therefore the number of columns in the Dixon matrix as determined after this perturbation is strictly less than the number determined by Theorem 4.1.5 if $\epsilon = (0, 0, \ldots, 0)$ (by an amount equal to the number of boundary integral points which moved out of the polytope). However, to derive asymptotic complexities, we will approximate the number of integral points inside any convex polytope by its volume. It can easily be see that even though perturbation removes some integral points from the polytope, it does *not* alter the volume of the polytope. Therefore, the asymptotic analysis and bounds derived as a consequence of Theorem 4.1.5 are unaffected whether or not one takes into account the perturbation. That is why we use the following Corollary in the following bound derivations:

Corollary 4.1.6 *For an unmixed set, $\mathcal{F} \subset \mathbb{Q}[X]$, of $n+1$ polynomials,*

$$\textit{Number of columns in the Dixon matrix} \;\leq\; \left| \sum_{i=0}^{n} \pi_i \left(\mathcal{N}(\mathcal{F}) \right) \cap \mathbb{Z}^n \right|.$$

Proof : Follows by setting $\epsilon = (0, 0, \ldots, 0)$ in Theorem 4.1.5. □

It should be pointed out that when exact analysis is required for *particular* examples (for instance, when one needs to determine precisely the set of monomials which label the columns of the Dixon matrix of a certain specialized set of polynomials), using the stronger version gives a more accurate picture of the Dixon formulation, as is demonstrated by the following example.

Example: Let $\mathcal{F}$ consist of three polynomials, p_1, p_2 and p_3 where

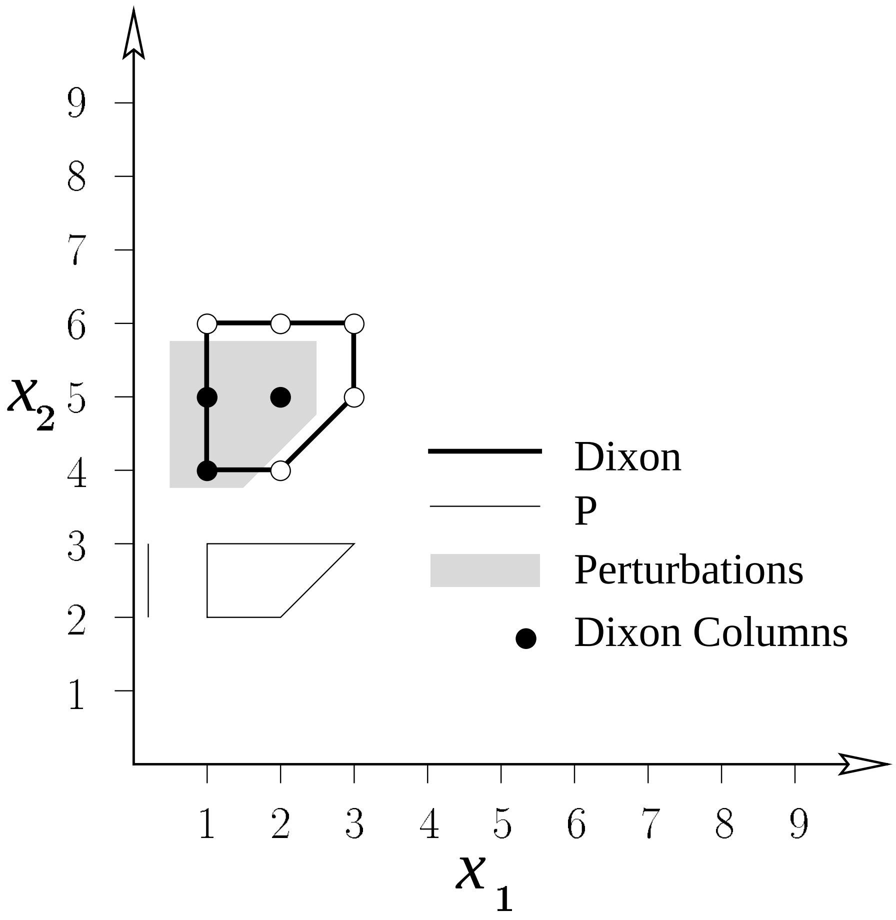

Figure 4.1: Polytope (w.r.t. X) of the Dixon Polynomial of a 3-polynomial system

$$p_i = a_i \left(x_1 x_2^2 - x_1 x_2^3 \right) + b_i x_1^2 x_2^2 + c_i x_1^2 x_2^3 + d_i x_1^3 x_2^3$$

Using the traditional construction methods described in the previous chapters, one obtains a Dixon polynomial δ for which:

$$V_X(\delta) = [x_1^2 x_2^5, x_1 x_2^5, x_1 x_2^4].$$

This means that there are three columns in the Dixon matrix. The number of columns, and indeed, the whole $V_X(\delta)$ vector can alternatively be determined using techniques outlined in this chapter as follows.

The Newton polytope of the system $\mathcal{F}$ (see Figure 4.1 for reference) is:

$$\mathcal{N}(\mathcal{F}) = conv\{(1,2),(2,2),(3,3),(1,3)\} = \pi_0(\mathcal{N}(\mathcal{F})).$$

Hence, $\pi_1(\mathcal{N}(\mathcal{F})) = conv\{(0,2),(0,3)\}$ and $\pi_2(\mathcal{N}(\mathcal{F})) = \{(0,0)\}$. It can easily be verified that

$$\sum_{i=0}^{2} \pi_i(\mathcal{N}(\mathcal{F})) = conv\{(1,4),(2,4),(3,5),(3,6),(1,6)\}.$$

This Minkowski sum is also drawn in Figure 4.1.

To apply Theorem 4.1.5, we perturb this summation by a *sufficiently* small amount towards the negative x_1 as well as x_2 axes. The perturbed region is shown as the shaded area in the figure. The set of integral points inside this perturbed region is found to be $\{(1,4),(1,5),(2,5)\}$. Hence the number of columns in the Dixon matrix

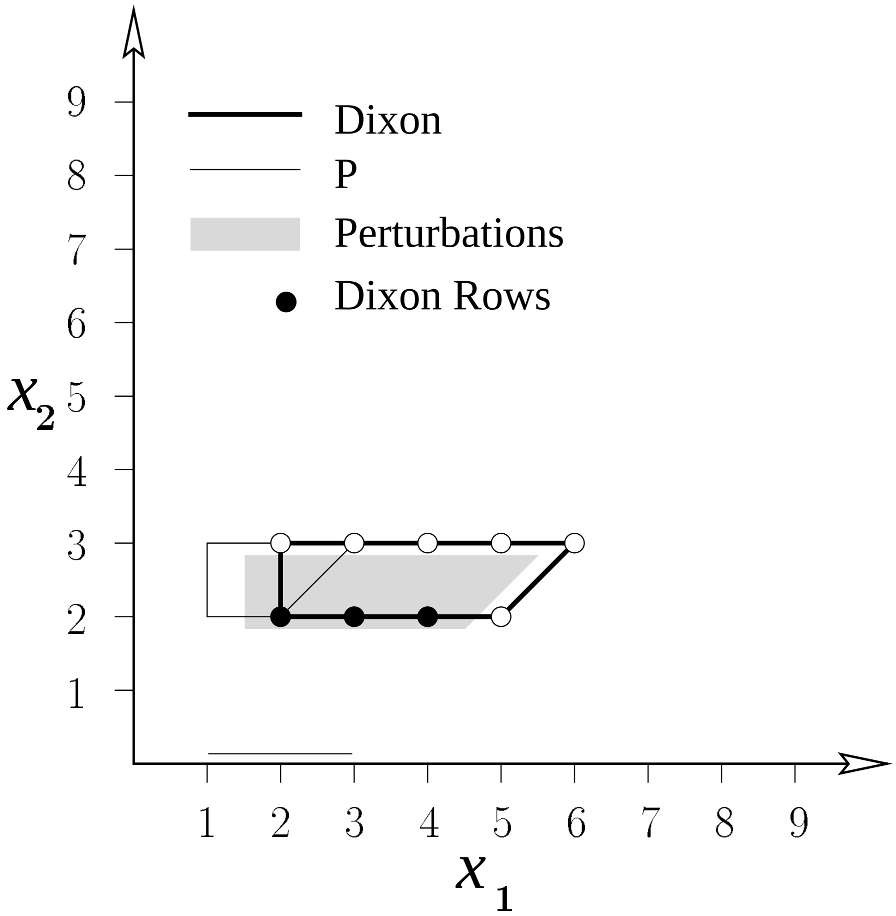

Figure 4.2: Polytope (w.r.t. $\bar{X}$) of the Dixon Polynomial of a 3-polynomial system

must be less than or equal to 3, which was exactly what was found using traditional construction. Moreover, the terms in x_is that the integral points in the shaded region represent, namely $x_1x_2^4, x_1x_2^5$ and $x_1^2x_2^5$, are exactly the monomials which label the columns of the Dixon matrix (namely, $V_X(\delta_P)$).

Likewise, if one wishes to determine the number of rows, or the monomials labeling the rows of the Dixon matrix, then one can compute a similar Minkowski sum by taking projections in the *reverse* order. In this case, one would add the Newton polytopes, $\mathcal{N}(\mathcal{F})$ and $conv(\{(3,0),(1,0)\})$ resulting in $conv(\{(2,2),(2,3),(5,2),(6,3)\})$.

Then slightly perturb this resulting Newton polytope in the negative x_1 and x_2 direction and look at the set of integral points inside the perturbed polytope. This construction is shown in Figure 4.2 and reveals that the rows of the Dixon matrix are also bounded by 3, and labeled by monomials $\bar{x}_1^2\bar{x}_2^2, \bar{x}_1^3\bar{x}_2^2$ and $\bar{x}_1^4\bar{x}_2^2$. This can be confirmed from the Dixon matrix constructed earlier. □

4.1.1 Mixed Systems

Our results have been proven only for unmixed systems. Derivation of tight bounds for the case of mixed systems seems much more difficult. This difficulty, though it did not seem easy for us to surmount, is encountered only at the first step of our proofs. Since each polynomial in the input system has a different polytope, it becomes difficult to derive a compact expression for the Newton polytope of the determinant of Cancellation matrix, which is the first step in Lemma 4.1.4. However, if such an expression *can* be derived, the remaining steps of our proof, i.e., those in Theorem 4.1.5 will still go through. So, the obstacle in deriving bounds for mixed systems is to derive an expression (similar to the one in Lemma 4.1.4) relating the Newton polytope of the determinant of the Cancellation matrix to the input Newton polytopes, which in this case are all different.

In the meantime, we deal with mixed systems by approximating them to unmixed systems. This can be achieved by assuming that the polytope of resulting unmixed system is the union of Newton polytopes of all polynomials in the mixed input system. In general the resulting polytopes can be quite large, hence the bounds obtained for the mixed systems in this fashion will not be very tight. However, some heuristics can be used to compute tighter bounds. One such heuristic we have used is when working with toric varieties. If one is only interested in conditions for solvability in the toric variety, then none of the variables can ever take the value 0 on all valid common solutions of the system. In other words, the polynomials of the input system can be divided by any monomial in X, without changing the conditions for existence of its common solutions. In terms of polytopes, this means that we are allowed to translate the Newton polytopes of the input polynomials. This gives the following nice heuristic to derive tight bounds on the size of the Dixon matrix, even when the system is mixed: Translate the Newton polytopes of input polynomials until the volume of the convex hull of their union is the smallest. Then approximate the given system by an unmixed system whose Newton polytope is this convex hull of the union of the translated polytope.

4.1.2 Summary

We believe this is the first time that a relationship between the Dixon matrix and the input Newton polytopes has been derived. As will be shown in the remainder of this thesis, our theorem will permit derivation of tight bounds on the size of the Dixon matrix and the time complexity of resultant computation using Dixon formulation. The stronger version of our theorem, which involves perturbation, precisely defines the structure of the Dixon polynomial and the Dixon matrix in terms of the Newton polytopes of the input polynomial. While methods based on Dixon's original formulation construct the Dixon matrix using algebraic methods of polynomial arithmetic, our results of this section provide a *computational geometric alternative* to construct the Dixon matrix.

4.2 Size of the Dixon Matrix

The relationship between the Newton polytopes of the input polynomials and their Dixon matrix can be exploited to derive tight bounds on the size of the Dixon matrices for different classes of polynomial systems. We demonstrate this by deriving tight bounds on the size of the Dixon matrices of *multi-homogeneous* polynomial systems in this section.

Multi-homogeneous systems form an important subclass which closely (possibly under a variable transformation) approximate polynomial systems appearing in many applications such as vision and robotics [14, 36]. They have been extensively studied in the context of homotopy methods by Morgan and Sommese [36] and in the context of sparse resultant formulation by Sturmfels and Zelevinsky [41] as well as by Canny and Emiris [14]. A multi-homogeneous polynomial system is defined as follows:

Definition 4.2.1 *A set of $n+1$ polynomials, where $n = \sum_{i=1}^{r} l_i$, is said to be* **multi-homogeneous of type** $(\mathbf{l_1}, \ldots, \mathbf{l_r}; \mathbf{d_1}, \ldots, \mathbf{d_r})$ *if it is unmixed and its variable set can be partitioned into r disjoined subsets, the i^{th} set having l_i variables, and the total degree of each polynomial in the i^{th} variable set is d_i.*

Polynomial systems arising in numerous applications are often either multi-homogeneous systems themselves, or can be approximated by one. For instance, the n-degree system discussed earlier is actually multihomogeneous systems of type $(1, \ldots, 1; d_1, \ldots, d_n)$. On the other hand, a polynomial of total degree d in n variables is a multihomogeneous of type $(n; d)$. A *multi-graded* system discussed in [14] is one in which for each $1 \leq i \leq r$, either $l_i = 1$ or $d_i = 1$. Often polynomial systems arising in applications can be *approximated* by multi-homogeneous systems. By approximated, we mean that the given polynomial system can be obtained from a multi-homogeneous system by removal of certain terms. In cases where this approximation is *close*, the results of multi-homogeneous systems also provide tight bounds on size of the Dixon matrix of the given polynomial system. As examples of specific real applications, the Stewart platform kinematics problem (see Chapter 7) [14] can be formulated as a problem of eliminating 6 variables from a multi-homogeneous system of type $(1,1,4;2,2,2)$. The camera motion from point matches problem (see Chapter 7) of computer vision approximates to an elimination of 5 variables from 6 polynomials of type $(2,3;1,1)$.

The reader should be cautioned that despite what the name suggests, polynomials in a multi-homogeneous system are *not* homogeneous themselves.

Since multi-homogeneous systems form an important subclass of polynomial systems, it becomes important to tightly bound the size of its resultant matrices. In special extreme cases such as n-degree systems, bounds based on the total degree can be tight and useful. For instance in Section 3.3.2, the following bound was derived for an n-degree system.

Theorem 4.2.2 *The size of the Dixon matrix of an n-degree polynomial set with $n+1$ polynomials in n variables, is exactly $n!\ \prod_{i=1}^{n} d_i$.*

However, for arbitrary multi-homogeneous systems, analysis based only on the total degrees of the polynomials, of the kind in Section 3.3, usually yields very loose bounds. In this section, we will demonstrate how the relationship between the Dixon matrix and the Newton polytopes of the input polynomial systems which we developed in the previous section can be exploited to derive optimal bounds on the sizes of the Dixon matrices of multi-homogeneous systems. In the first theorem below, we derive the volume of the Newton polytope of a general multi-homogeneous polynomial.

Theorem 4.2.3 *The volume of the Newton polytope $\mathcal{N}$ of a multi-homogeneous polynomial of type $(l_1, \ldots, l_r; d_1, \ldots, d_r)$ is:*

$$vol\,(\mathcal{N}) = \prod_{i=1}^{r} \frac{d_i^{l_i}}{l_i!}.$$

Proof : Let $\mathcal{F}$ be a polynomial system of type $(l_1, \ldots, l_r;\ d_1, \ldots,\ d_r)$. First a formula for the volume of $\mathcal{N}(\mathcal{F}) \subset \mathbb{R}^n$ is derived, then a formula for the volume of the Minkowski sum of successive projections of $\mathcal{N}(\mathcal{F})$ is developed. All the mathematics in this proof is an application of high-school calculus to compute volumes and areas of n-dimensional figures.

Let x_{ij} be the j^{th} variable in the i^{th} partition of the variable set of $\mathcal{F}$. Let y_{ij} range over the exponent of x_{ij}. We view terms in x_{ij}s as points corresponding to their exponent vectors in the n-dimensional space with axes labeled by y_{ij}s.

$\mathcal{N}(\mathcal{F})$ is the Minkowski sum of r l_i-dimensional simplices L_i, constructed as follows. L_i is the convex hull of l_i edges, where for each $1 \le j \le l_i$, there is the edge of length d_i from the origin along the y_{ij}-axis in positive direction. Since these simplices are independent of each other, the n-dimensional volume enclosed by $\mathcal{N}(\mathcal{F})$, say U, is the product of the l_i-dimensional volumes, U_i, of L_i given by:

$$U_i \;=\; \int_0^{d_i} \int_0^{d_i - y_{i1}} \int_0^{d_i - y_{i1} - y_{i2}} \cdots \int_0^{d_i - \sum_{j=1}^{l_i - 1} y_{ij}} dy_{il_i}\, dy_{il_i - 1} \cdots dy_{i1}$$

$$
\begin{aligned}
&= \frac{d_i^{l_i}}{l_i!}, \\
U &= \prod_{i=1}^{r} U_i = \prod_{i=1}^{r} \frac{d_i^{l_i}}{l_i!}.
\end{aligned}
$$

$\square$

Along the same lines as the proof of the previous theorem, the following theorem derives the precise bound on the size of the Dixon matrix of an arbitrary multi-homogeneous system.

Theorem 4.2.4 *The size of the Dixon matrix of a multi-homogeneous polynomial set of type $(l_1, \ldots, l_r;\ d_1, \ldots,\ d_r)$ with $n+1$ polynomials, where $n = \sum_{i=1}^{r} l_i$ is the total number of variables, is*

$$
\mathcal{O}\left(\frac{1}{n+1}\prod_{i=1}^{r}\frac{\left(d_i\left(\sum_{j=1}^{i} l_j + 1\right)\right)^{l_i}}{l_i!}\right).
$$

Proof : The volume of the Minkowski sum of successive projections of the input Newton polytopes, $\mathcal{Q} = \sum_{k=0}^{n} \pi_k\left(\mathcal{N}(\mathcal{F})\right) \subset \mathbb{R}^n$ can again be computed in the same fashion as the previous theorem. The polytope $\mathcal{Q}$ is the Minkowski sum of r l_i-dimensional polytopes N_i, constructed as follows. For $1 \leq i \leq r$, let M_i be the convex hull of l_i edges, where for each $1 \leq j \leq l_i$, there is the edge of length $\left(\sum_{j=1}^{i} l_j\right) d_i$ from the origin along the y_{ij}-axis in positive direction. N_i is obtained by truncating M_i by l_i half-spaces, $y_{ik} \leq \left(\sum_{j=1}^{i-1} l_j + k\right) d_i$, $1 \leq k \leq l_i$. M_i is L_i scaled by a factor of $\sum_{j=1}^{i} l_j$ along each of its l_i dimensions. This scaling occurs because of the Minkowski sum of the successive projections of $\mathcal{N}(\mathcal{F})$ under all the variables before the i^{th} partition. N_i is constructed by truncation of M_i to account for the successive projection of variables *within* the i^{th} partition. The volume V of $\mathcal{Q}$ is the product of the volumes V_i of each N_i, which can be derived to be:

$$
\begin{aligned}
V_i &= \int_0^{\left(\sum_{j=1}^{i-1} l_j+1\right) d_i} \int_0^{\left(\sum_{j=1}^{i-1} l_j+2\right) d_i - y_{i1}} \int_0^{\left(\sum_{j=1}^{i-1} l_j+3\right) d_i - y_{i1} - y_{i2}} \\
&\quad \cdots \int_0^{\left(\sum_{j=1}^{i} l_j\right) d_i - \sum_{j=1}^{l_i-1} y_{ij}} dy_{il_i} \cdots dy_{i1} \\
&= \frac{d_i^{l_i} \left(\sum_{j=1}^{i-1} l_j + 1\right) \left(\sum_{j=1}^{i} l_j + 1\right)^{l_i-1}}{l_i!}, \\
V &= \prod_{i=1}^{r} V_i = \frac{1}{n+1} \prod_{i=1}^{r} \frac{\left(d_i \left(\sum_{j=1}^{i} l_j + 1\right)\right)^{l_i}}{l_i!}.
\end{aligned}
$$

In Theorem 4.1.5, the size of the Dixon matrix was derived to be the number of integral points inside the convex polytope $\sum_{i=0}^{n} \pi_i\left(\mathcal{N}(\mathcal{F})\right)$. In [13], it is proved that the number of integral points inside a convex polytope is of the same order as the volume of that polytope, and the theorem follows. □

Our previous bounds on the size of the Dixon matrix in [29] were loose because the analysis had not taken the sparsity of the input polynomials into consideration. On the other hand, the bound on the size of the Dixon matrix of a multi-homogeneous system in Theorem 4.2.4 is tight. In fact, if the polynomial system is generic, i.e., all the coefficients are symbolic and independent of each other, then this bound is exact. It was possible to derive such an exact bound only because Theorem 4.1.5 precisely describes the support of the Dixon polynomial by exploiting the sparsity in the input polynomials in terms of the structure of their Newton polytopes.

The formula in Theorem 4.2.4 looks quite cumbersome. A simplified expression which exploits the order notation can be derived in terms of the *mixed volume* as follows.

Given an unmixed set $\mathcal{F} \subset \mathbb{Q}[X]$, its n**-fold mixed volume**, denoted $mvol(\mathcal{F})$, is $mvol(\mathcal{F}) = n!\ vol(\mathcal{N}(\mathcal{F}))$, where $vol(\mathcal{N}(\mathcal{F}))$ is the n-dimensional volume of the Newton polytope of $\mathcal{F}$. Let the number of partition variables $l_i = \mathcal{O}(l)$, and the partition degree $d_i = \mathcal{O}(d)$, and consequently the total number of variables, $n = \mathcal{O}(r\,l)$. The following corollary expresses the size of the Dixon matrix of a multi-homogeneous polynomial system in terms of its mixed volume.

Corollary 4.2.5 *The size of the Dixon matrix of a multi-homogeneous system $\mathcal{F}$ is*

$$\mathcal{O}\left(\frac{(r!)^l\ l^n}{(n+1)!}\right) mvol\,(\mathcal{F})\,.$$

Proof : For unmixed systems, it can be shown that $mvol\,(P) = n!vol\,(P)$. From the Theorem 4.2.3, $vol\,(P) = \left(\frac{d^l}{l!}\right)^r$. Since $n = rl$, it follows that $mvol\,(P) = n!\frac{d^n}{l!^r}$. Now, the expression for the size, S_d, of the Dixon matrix from Theorem 4.2.4 can be simplified as follows:

$$\begin{aligned}
S_d &= \frac{1}{n+1}\prod_{i=1}^{r}\frac{d^l\,(il+1)^l}{l!} \\
&= \frac{1}{n+1}\frac{d^{rl}}{l!^r}\left(\prod_{i=1} r\,(il+1)\right)^l \\
&= \frac{1}{(n+1)!}\frac{n!d^n}{l!^r}\left(\prod_{i=1} ril\right)^l \\
&= \frac{1}{(n+1)!}mvol\,(P)\,r!^l l^{rl}
\end{aligned}$$

$$= \frac{r!^l l^n}{(n+1)!} mvol\,(P)$$

$\square$

Now that we have demonstrated that Dixon formulation does take the sparsity of the input polynomials into account, and derived tight bounds on the size of the Dixon matrix based on this relationship, we will next compare these results to sparsity considerations in other formulations. In the next section, we discuss the similarities and differences between the sparsity exploitation as seen in the Dixon formulation with other formulations. We also derive bounds on the sizes of the resultant matrices in other formulations and compare them to the size of the Dixon matrix derived in this section.

4.3 Sparsity Considerations in Other Formulations

4.3.1 Macaulay Formulation

Macaulay's method was probably the first method for computing the resultant of multivariate polynomials. As stated earlier, this method also constructs a matrix, the Macaulay matrix, whose determinant is a projection operator. The size of the Macaulay matrix in the most general case can be computed as follows (see [27]):

Theorem 4.3.1 *Let $\mathcal{F} = \{p_1, \ldots, p_{n+1}\}$ be a set of $n+1$ polynomial in n variables such that the total degree of polynomial p_i is d_i. Let $d_{\mathcal{F}} = \sum_{i=1}^{n+1} d_i - n$. The size of the Macaulay matrix of $\mathcal{F}$ is exactly $\binom{d_{\mathcal{F}}+n}{n}$.*

So, if the total degree of each input polynomials is $\mathcal{O}(d)$, then the size of the Macaulay matrix is $\mathcal{O}\left(\frac{(n+1)^n d^n}{n!}\right)$. It is clear from these formulae that the total degree of input polynomials completely determines the size of their Macaulay matrix, no matter how many terms are missing from the polynomials. If the polynomials of a system $\mathbf{M}$ have a larger total degree than polynomials of another system $\mathbf{N}$, then $\mathbf{M}$ has a larger Macaulay matrix, even though the volumes of the Newton polytopes of $\mathbf{N}$ may be larger than those of $\mathbf{M}$.

Corollary 4.1.5 shows that, for the Dixon matrix, this is not the case at all. In fact, the size of the Dixon matrix does not depend on the total degree of the polynomials, only on their Newton polytopes. Sizes of the Dixon matrices of two different polynomial systems are the same as long as their Newton polytopes have the same shape, no matter what the total degrees of the polynomials.

In other words, unlike the Macaulay formulation, the Dixon formulation *does* consider the sparsity in the input polynomials and implicitly exploits the structure of the Newton polytopes of input polynomials to construct smaller matrices.

4.3.2 Sparse Resultant Formulation

The sparse resultant method, presented in [15] computes a projection operator of a polynomial system while exploiting the Newton polytopes of the polynomials to construct Macaulay-like (but smaller) matrices. In [14], many results about sparse resultant matrices are given. In particular, two algorithms for constructing the sparse resultant matrix are described. The first is the subdivision algorithm [[14], section 3.1] which constructs resultant matrices whose size is as follows.

Theorem 4.3.2 *The size of the sparse resultant matrix (constructed using the subdivision algorithm) of an unmixed set, $\mathcal{F} \subset \mathbb{Q}[X]$, of $n+1$ polynomials is*

$$\mathcal{O}\left(vol\left(\sum_{i=0}^{n} \mathcal{N}(\mathcal{F})\right)\right).$$

Note the absence of i inside this summation. Therefore by $\sum_{i=0}^{n} \mathcal{N}(\mathcal{F})$, we simply mean to add $\mathcal{N}(\mathcal{F})$ to itself n times.

As is evident from Theorems 4.3.2 and 4.1.5, there is a lot of similarity in the way sparsity is exploited in the Dixon and Sparse resultant formulations. While the columns of the sparse resultant matrix are labeled by monomials corresponding to the integral points inside the Minkowski sum of the Newton polytopes, columns of the Dixon matrix are labeled by integral points inside the Minkowski sum of *successive projections* of Newton polytopes.

A natural question that arises is – Which monomials can be introduced into the polynomials in $\mathcal{F}$ while keeping the bound on the size of their Dixon matrix, determined by corollary 3.11, constant? It is evident from the corollary, and the theorem preceeding it, that any monomial which does not change the Newton polytope of $\mathcal{F}$ can be introduced into the polynomials of $\mathcal{F}$ without exceeding the bound determined by the corollary. In other words, any monomial which lies within the volume determined by the Newton polytope of $\mathcal{F}$ can be introduced.

But are these the only monomials whose introduction will keep the bound unchanged? The answer to this is also in the affirmative. This is because the introduction of any monomial outside the Newton polytope will definitely change one of the summands of the expression in Theorem 4.1.5. Introduction of monomials lying outside, but close to, the original Newton polytope, $\mathcal{N}(\mathcal{F})$, may not change any of the projections $\pi_i(\mathcal{N}(\mathcal{F}))$, for $1 \leq i \leq n$, but they *will* change one of the summands in the Minkowski sum, namely, $\pi_0(\mathcal{N}(\mathcal{F})) = \mathcal{N}(\mathcal{F})$. Since changing any of the summands of the expression in Theorem 4.1.5 will expand the shape of the total Minkowski sum, and increase the volume of this sum, it will also increase the size of the Dixon matrix. So, *the set of monomials which can be introduced into the polynomials of an unmixed set of polynomials without changing the bound on the size of their Dixon matrix is exactly the set of monomials lying in or on the Newton polytope of the polynomials.*

We can ask the same question in the context of sparse resultant formulation, namely - Which monomials can be introduced into the polynomials in $\mathcal{F}$ while keeping the bound on the size of their sparse resultant matrix, determined by Theorem 4.3.2, constant? It follows from Theorem 3.12 that the size of the sparse resultant matrix depends directly on the volume of the Newton polytope of $\mathcal{F}$. So, the answer in this case would also be exactly the same as in the Dixon case.

This observation is important enough to be stated precisely.

Fact 4.3.3 *In the Dixon and the sparse resultant formulations, the set of monomials which can be introduced into the polynomials of an unmixed set $\mathcal{F} \subset \mathbb{Q}[X]$, while keeping the size of the corresponding resultant matrices constant, is exactly the same. In particular, it is exactly the set of monomials which lie inside the Newton polytope of $\mathcal{F}$. On the other hand, the set of monomials which keep the size of the Macaulay matrix constant is simply the set of monomials whose total degree is not larger than that of $\mathcal{F}$; this set has no relationship with the volume enclosed by the Newton polytope*

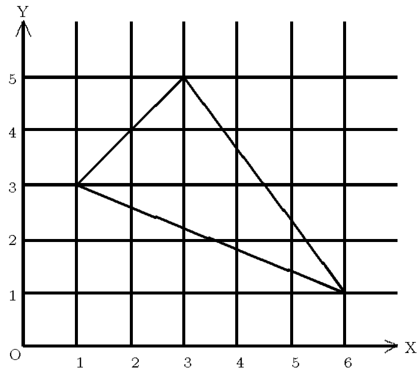

Figure 4.3: Newton polytope for example 3.

of $\mathcal{F}$.

This relationship will be further illustrated by the following example:

Example: Consider an unmixed set $\mathcal{F}$ of three polynomials in two variables whose support is

$$supp(\mathcal{F}) = \{(1,3),(2,4),(3,5),(4,3),(4,2),(5,2),(6,1)\}.$$

The Newton polytope of this system is shown in Figure 4.3.

Dixon matrix of this system is 20×20, the sparse resultant matrix is 50×50, and the Macaulay matrix is 276×276. If one adds any monomial within the Newton polytope, shown in Figure 4.3, to the polynomials of $\mathcal{F}$, then none of the three matrices' sizes increase. In other words, the introduction of any one or more monomials from the set

$$\{(2,3),(3,3),(3,4)\}$$

to the polynomials of $\mathcal{F}$ does not change the sizes of any of the three resultant matrices. However, if one adds the monomial $(3,2)$, which lies *outside* the Newton polytope (see Figure 4.3), then the size of the Dixon matrix increases to 21×21 and the sparse resultant matrix becomes 57×57. Since the total degree of this monomial (5) is still less than the total degree of the polynomials in $\mathcal{F}$ (8), the size of the Macaulay matrix remains unchanged. But, if one adds the monomial $(5,4)$, which not only lies outside the Newton polytope but also has a total degree (9), larger than the total degree of $\mathcal{F}$ (8), the sizes of all three resultant matrices increase. The Dixon matrix becomes 23×23, sparse resultant matrix becomes 75×75 and the Macaulay matrix becomes 351×351. □

We have described the extent to which, and the fashion in which the three formulations exploit the sparsity in input polynomials to construct smaller resultant matrices. These discussions were based on theorems which precisely relate the sizes of the resultant matrices to the structure of input polynomials (Theorems 4.1.5, 4.3.2, 4.3.1). However, we have not directly compared the sizes of the three resultant matrices. We carry out this comparison next.

4.3.3 Comparison of the Resultant Matrix Sizes

As was mentioned, the size of the Dixon matrix of an unmixed system is bounded by the number of integral points inside the Minkowski sum of successive projections of the input Newton polytope, whereas the size of its sparse resultant matrix is bounded by the number of integral points inside the Minkowski sum of the input Newton polytope $n+1$ times. Since the projections of Newton polytopes with positive n-dimensional volume are of successively lower dimensions than the Newton polytopes themselves, the Dixon matrix is smaller than the sparse resultant matrix. In this section we will precisely derive the factor by which the Dixon matrix is smaller than sparse and Macaulay resultant matrices.

Below, we recapitulate the formulas for the sizes of the Dixon, sparse and Macaulay matrices for multi-homogeneous systems. In the following, we denote the sizes of the Dixon matrix, sparse resultant matrix, and the Macaulay matrix of $\mathcal{F}$ by S_d, S_s and S_m respectively.

Theorem 4.3.4 *Let $\mathcal{F}$ be a multi-homogeneous polynomial set of type $(l_1, \ldots, l_r;$ $d_1, \ldots, d_r)$ with $n+1$ polynomials, where $n = \sum_{i=1}^{r} l_i$ is the total number of variables. Then,*

$$
\begin{aligned}
vol\,(\mathcal{N}\,(\mathcal{F})) &= \prod_{i=1}^{r} \frac{d_i^{l_i}}{l_i!}, \\
mvol\,(\mathcal{F}) &= n!\ vol\,(\mathcal{N}\,(\mathcal{F})) \\
S_d &= \mathbf{O}\left(\frac{\prod_{i=1}^{r} \left(\sum_{j=1}^{i} l_j + 1\right)^{l_i}}{n+1}\ vol\,(\mathcal{N}\,(\mathcal{F})) \right) \\
S_s &= \mathbf{O}\left((n+1)^n\ vol\,(\mathcal{N}\,(\mathcal{F}))\right) \\
S_m &= \mathbf{O}\left(\frac{(n+1)^n \left(\sum_{i=1}^{r} d_i\right)^n}{n!} \right)
\end{aligned}
$$

The following Theorem compares the asymptotic sizes of the three resultant matrices.

Theorem 4.3.5 *Let $\mathcal{F}$ be a multi-homogeneous polynomial set of type $(l_1, \ldots, l_r;$ $d_1, \ldots, d_r)$ with $n+1$ polynomials, where $n = \sum_{i=1}^{r} l_i$ is the total number of variables. For $1 \le i \le r$, let $l_i = \mathbf{O}(l)$ and $d_i = \mathbf{O}(d)$, and consequently, $n = \mathbf{O}(r\,l)$. Then,*

1. *for asymptotically large r and l, $S_s = \mathbf{O}(n\ e^n)\ S_d$ and $S_m = \mathbf{O}(1)\ S_s$ and,*

2. *for small l (in particular for $l=1$) but asymptotically large r, $S_s = \mathbf{O}(n\ e^n)\ S_d$ and $S_m = \mathbf{O}(e^n)\ S_s$.*

Proof : The proof follows from the previous theorem using Stirlings approximation to the factorial. For asymptotically large r, and arbitrary l, the expression relating S_s and S_d is derived as follows.

$$\begin{aligned}\frac{S_s}{S_d} &= \mathbf{O}\left(\frac{(rl+1)\,(rl+1)^{rl}}{\prod_{i=1}^{r}(il+1)^{l}}\right) = \mathbf{O}\left((rl+1)\left(\prod_{i=1}^{r}\frac{rl+1}{il+1}\right)^{l}\right) = \mathbf{O}\left(rl\left(\prod_{i=1}^{r}\frac{r}{i}\right)^{l}\right) \\ &= \mathbf{O}\left(rl\left(\frac{r^r}{r!}\right)^{l}\right) = \mathbf{O}\left(rl\,(e^r)^l\right) = \mathbf{O}\left(n\ e^n\right).\end{aligned}$$

This proves the relationship between S_s and S_d for both the items in the statement of the theorem. The relationship between S_m and S_s is proven as follows:

$$\frac{S_m}{S_s} = \mathbf{O}\left(\frac{(n+1)^n\,(rd)^n}{n!\ (n+1)^n\,vol\,(\mathcal{N}\,(\mathcal{F}))}\right) = \mathbf{O}\left(\frac{(n+1)^n\,(rd)^n\,(l!)^r}{n!\ (n+1)^n\,d^n}\right) = \mathbf{O}\left(\frac{r^n\,(l!)^r}{n!}\right) \tag{4.2}$$

Now, for item 1 in the theorem, i.e., for asymptotically large r and l, we continue from equation 4.2 as follows:

$$\frac{S_m}{S_s} = \mathbf{O}\left(\frac{r^n\left(l^l\right)^r e^n}{n^n\,(e^l)^r}\right) = \mathbf{O}\left(\frac{r^n l^n e^n}{n^n e^n}\right) = \mathbf{O}\left(\frac{n^n e^n}{n^n e^n}\right) = \mathbf{O}(1).$$

For item 2 in the theorem, ie., for asymptotically large r, but small l, note that $n = \mathbf{O}(r)$. The proof follows from equation 4.2:

$$\frac{S_m}{S_s} = \mathbf{O}\left(\frac{n^n\,(1!)^r}{n!}\right) = \mathbf{O}\left(\frac{n^n}{n!}\right) = \mathbf{O}\left(e^n\right).$$

$\square$

For a set of multi-homogeneous polynomial equations, if the number of partitions, r, is large but if the number of variables in each partition is small, Dixon matrix is smaller than the sparse resultant matrix by a factor, exponential in the total number of variables, n, in all the partitions. Sparse resultant matrix is, in turn, smaller than the Macaulay matrix by a factor, also exponential in n. If the number of variables in each partition, l, is also large, then Dixon matrix is still smaller than the sparse resultant matrix by a factor exponential in n, but the sparse resultant matrix becomes almost the same size as the classical Macaulay matrix!

An important relationship is that between the size of the resultant matrices and the n-fold mixed volume of the polynomial system. We have already mentioned before that the Macaulay matrix is based on the Bezout bound and has no relationship with the n-fold mixed volume. However, the sparse resultant was formulated as an attempt to exploit the structure of the Newton polytopes while building the resultant matrix, so it has a relationship with the n-fold mixed volume. The surprising result is the relationship of the Dixon matrix to the n-fold mixed volume. The following theorem expresses the relationships of the sizes of the Dixon and sparse resultant matrices of a multi-homogeneous polynomial set with its n-fold mixed volume.

Theorem 4.3.6 *Let $\mathcal{F}$ be a multi-homogeneous polynomial set of type $(l_1, \ldots, l_r;$ $d_1, \ldots, d_r)$ with $n+1$ polynomials, where $n = \sum_{i=1}^{r} l_i$ is the total number of variables. For $1 \le i \le r$, let $l_i = \mathbf{O}(l)$ and $d_i = \mathbf{O}(d)$, and consequently, $n = \mathbf{O}(r\,l)$. Then, for asymptotically large r,*

$$\begin{aligned} S_d &= \mathbf{O}\left(\frac{1}{n+1}\right) mvol(\mathcal{F}) \text{ and} \\ S_s &= \mathbf{O}\left(e^{n+1}\right) mvol(\mathcal{F}). \end{aligned}$$

Proof : From Theorem 4.3.4 for multi-homogeneous polynomials, we have

$$\begin{aligned} \frac{S_d}{mvol\,(\mathcal{F})} &= \frac{\prod_{i=1}^{r}(i\,l+1)^l}{(rl+1)\,(r\,l)!} = \mathbf{O}\left(\frac{e^{r\,l}\prod_{i=1}^{r}(i\,l+1)^l}{(rl+1)\,(rl)^{r\,l}}\right) \\ &= \mathbf{O}\left(\frac{e^{r\,l}}{rl+1}\left(\prod_{i=1}^{r}\frac{il+1}{rl}\right)^l\right) = \mathbf{O}\left(\frac{e^{r\,l}}{rl+1}\left(\prod_{i=1}^{r}\frac{i}{r}\right)^l\right) \\ &= \mathbf{O}\left(\frac{e^{r\,l}}{rl+1}\left(\frac{r!}{r^r}\right)^l\right) = \mathbf{O}\left(\frac{e^{rl}}{rl+1e^{rl}}\right) = \mathbf{O}\left(\frac{1}{n+1}\right). \end{aligned}$$

The result for the sparse resultant matrix is easily obtained from Theorem 4.3.2:

$$\frac{S_s}{mvol\,(\mathcal{F})} = \mathbf{O}\left(\frac{(n+1)^n}{n!}\right) = \mathbf{O}\left(\frac{(n+1)^{n+1}}{(n+1)!}\right) = \mathbf{O}\left(e^{n+1}\right).$$

□

This is a surprising result that for asymptotically large partitions in a multi-homogeneous set of polynomials, the size of the Dixon matrix is almost the same as the n-fold mixed volume. On the other hand, the size of the sparse resultant matrix is the n-fold mixed volume times an exponential factor in the total number of variables.

Instead of the total number of variables, the sizes of Dixon and sparse resultant matrices can also be compared in terms of the number of partitions. It can be easily derived that [[14], Theorem 2.2.5]:

Theorem 4.3.7 *The size of the sparse resultant matrix (constructed using the subdivision algorithm) of an unmixed set, $\mathcal{F} \subset \mathbb{Q}[X]$, of $n+1$ polynomials is*

$$\mathcal{O}\left(\frac{(n+1)^n}{n!} mvol(\mathcal{F})\right).$$

Theorem 4.3.7 and Corollary 4.2.5 can be used to compare the sizes of the Dixon and sparse resultant matrices of multi-homogeneous polynomial systems as follows.

Corollary 4.3.8 *For asymptotically large number of variables, the Dixon matrix of a multi-homogeneous system with r partitions and $\mathcal{O}(l)$ variables in each partition is smaller than its sparse resultant matrix (constructed using the subdivision algorithm) by a factor of $\mathcal{O}((r^r/r!)^l)$.*

Proof : We know from Corollary 4.2.5 that $S_d = \frac{l^n r!^l}{(n+1)!} mvol(P)$. Using Theorem 4.3.7, we get:

$$\begin{aligned}\frac{S_s}{S_d} &= \mathcal{O}\left(\frac{(n+1)^n}{n!}\frac{(n+1)!}{l^n r!^l}\right)\\ &= \mathcal{O}\left(\frac{(n+1)^{n+1}}{l^n r!^l}\right) = \mathcal{O}\left(\frac{n^n}{l^n r!^l}\right) = \mathcal{O}\left(\frac{(rl)^n}{l^n r!^l}\right)\end{aligned}$$

$$= \mathcal{O}\left(\frac{r^{rl}}{r!^l}\right) = \mathcal{O}\left(\left(\frac{r^r}{r!}\right)^l\right)$$

□

All of the above results have been derived assuming that the subdivision algorithm in [14] was used to construct the sparse resultant matrix. The second algorithm for constructing sparse resultant matrices is an incremental algorithm [[14], section 3.2] which uses a direction vector to construct still smaller matrices. For multi-graded systems, an optimal direction vector can be defined, and a sparse resultant matrix of size exactly $mvol(\mathcal{F})$ can be constructed using this algorithm. However, for multi-homogeneous systems which are not multi-graded, it is unclear how an optimal direction vector should be chosen [[14], p. 51]. The worst case bound on the size of the sparse resultant matrices constructed by the incremental algorithm is the same as that of the subdivision algorithm, and is given by Theorem 4.3.7.

Example: Consider again the three polynomial system $\mathcal{F}$ from the previous example with $\mathcal{N}(\mathcal{F}) = conv\{(1,2),(2,2),\ (3,3),(1,3)\}$. Refer to Figure 1 for the various polytopes discussed in this example. It can be easily verified that

$$\sum_{i=0}^{2} \mathcal{N}(\mathcal{F}) = conv\{(3,6),(6,6),(9,9),(3,9)\}.$$

Analogous to the result for the size of the Dixon matrix involving a slight perturbation mentioned previously, the exact size of the Sparse resultant matrix can also be computed by counting the number of integral points inside $\sum_{i=0}^{2} \mathcal{N}(\mathcal{F})$ after slightly perturbing it in the negative x_1 and x_2 axes. The set of integral points after this perturbation is

$$\{(3,6),(3,7),(3,8),(4,6),(4,7),(4,8),(5,6),(5,7),(5,8),(6,7),(6,8),(7,8)\}.$$

Hence the size of the Sparse resultant matrix is bounded by 12. In fact, the sparse resultant matrix constructed by a greedy variant of the subdivision algorithm is 11×11, and the sparse resultant matrix constructed by an implementation of the incremental algorithm is 9×9. Recall that the Dixon matrix was 2×3.

It should also be pointed out that since the Macaulay formulation disregards the sparsity completely and only takes the total degree of the polynomials (which here is 6) into account, the size of the Macaulay matrix for this system is 153×153. This can also be derived from the formula above as $d_M = 3 \times 6 - 2 = 16$, and the size of the Macaulay matrix is $\binom{16+2}{2} = 153$. $\square$

We conclude this section by taking a detailed look at yet another illustrative example. Even though this example is not representative of the asymptotic analysis above (simply because the number of variables and partitions in it are not asymptotically large), it underlines the fact that the Dixon matrix is far smaller than the sparse resultant matrix in practice also, not just in theory as demonstrated above.

Example: : Consider a multi-homogeneous system of type $(1,2,1;1,1,1)$. Its n-fold mixed volume is 12. The Dixon matrix for such a system is 12×12. The sparse resultant matrix is 230×230. So the sparse resultant matrix is $230/12 = 19.167$ times larger than the Dixon matrix.

One can derive a similar ratio directly from the formulas in Theorem 4.3.4 as follows. The values of various parameters here are: $l_1 = 1, l_2 = 2, l_3 = 1$, therefore,

$n = 4$. So, the ratio of the size of the sparse resultant matrix and the Dixon matrix can approximately be derived to be:

$$\frac{S_s}{S_d} = \frac{(n+1)^n\,(n+1)}{(l_1+1)^{l_1}\,(l_1+l_2+1)^{l_2}\,(l_1+l_2+l_3+1)^{l_3}} = \frac{5^5}{2\times 4^2\times 5} = 19.53125$$

So, we see that we get approximately the same ratio as we got by explicit construction of the matrices using a computer implementation of the algorithms. □

Nevertheless, smaller resultant matrices are not of much use unless they can be constructed in a small time. The next chapter deals with precisely that issue.

Chapter 5

Complexity of Computing Resultants

Constructing the Dixon Matrix

Up till this point, this thesis has provided tight bounds on the size of the Dixon matrix. This will enable us to derive tight bounds on the time complexity of computing the resultant once the Dixon matrix has been constructed. However, we have not yet dealt with the issue of constructing the Dixon matrix. The only procedure for Dixon matrix construction we outlined was the one in the classical Dixon formulation, where the determinant of the Cancellation matrix is expanded, then divided by $\prod_{i=1}^{n}(x_i - \bar{x}_i)$ to obtain the Dixon polynomial from which the Dixon matrix can be constructed. However, it is conceivable to exploit the relationship of the term structure of the Dixon polynomial to the input Newton polytopes for efficient construction of the Dixon matrix. This chapter explores this possibility, and develops an efficient algorithm which exploits the input Newton polytopes to construct the Dixon matrix faster, thus enabling us to relate the complexity of resultant computation using Dixon formulation to the structure of the input Newton polytopes.

The reason it can be expensive to construct the Dixon matrix by symbolically expanding the determinant of the cancellation matrix and then dividing it by

$\prod_{i=1}^{n}(x_i - \bar{x}_i)$ is that the entries of the cancellation matrix are multivariate polynomials. Symbolically expanding its determinant can give rise to huge intermediate expressions, resulting in inefficiency. On the other hand, since the term structure of the Dixon polynomial can be precisely determined using techniques from the previous chapter, it can also be exploited to construct the Dixon matrix fast using interpolation methods [45]. In particular, the precise bound on the number of terms in the Dixon polynomial which was developed in the previous chapter can be used to interpolate the Dixon polynomial using dense polynomial interpolation. Since entries of the Dixon matrix are simply the coefficients of the Dixon polynomial, the Dixon matrix can be constructed as fast as the Dixon polynomial can be interpolated. Since the number of terms in the Dixon polynomial depends on the volumes of the input Newton polytopes, so does the complexity of constructing the Dixon matrix.

In the next section, we will describe an algorithm which exploits the relationship between the input Newton polytopes and the Dixon polynomial to construct the Dixon matrix efficiently. We derive its complexity and show its dependence on the volumes of the input Newton polytopes. In the second section we use the bounds on the size of the Dixon matrix, and the steps required to construct it, to derive the total complexity of computing the resultant using our extension of the Dixon formulation. Our results in this section will demonstrate the dependence of the complexity of resultant computation using the Dixon formulation to the volumes of the input Newton polytopes. In the last section we compare our results to the complexity of computing resultants using the Macaulay and the sparse resultant formulations.

5.1 Construction of the Dixon Matrix

Multivariate interpolation works in stages; at each stage one variable is lifted [45]. The cost of each stage of interpolation depends on:

1. the number of terms in the polynomial when it is viewed as a polynomial in the variables which have already been lifted, and

2. the degree of the polynomial in the variable being lifted.

Since we have already shown that the number of terms in the Dixon polynomial depends on the volume of the input Newton polytopes, so does the complexity of interpolating it. We now give the details of the algorithm and derive its complexity.

Algorithm

Let $\mathcal{F}$ be the polynomial system whose Dixon matrix is to be constructed. Let $X = \{x_1, \ldots, x_n\}$ be the n variables. As was mentioned, since polynomials are interpolated one variable at a time, before attempting to interpolate a polynomial the order in which the variables will be lifted must be decided upon. Our derivations have shown that we achieve the best results by interpolating variables in the increasing order of their degrees. The polynomial we want to interpolate is the Dixon polynomial, which is a polynomial in $2n$ variables, X and $\bar{X} = \{\bar{x}_1, \ldots, \bar{x}_n\}$. It can be easily seen that the degree of the Dixon polynomial in the variables of X monotonically rises as one goes from x_1 to x_n. Moreover, from the structure of the cancellation matrix, it can also be concluded that the Dixon polynomial of an unmixed system has the same degree in x_i as in $\bar{x}^{n-i+1}$. That is why, in our algorithm, we will lift the $2n$ variables, namely X and $\bar{X}$, in the Dixon polynomial $\delta_{\mathcal{F}}$ in the order $x_1, \bar{x}_n, x_2, \bar{x}_{n-1}, \ldots, x_n, \bar{x}_1$, because the degree of the Dixon polynomial rises monotonically w.r.t. successive variables in this list.

Let the k^{th} variable in this sequence be denoted by v_k. So, $v_3 = x_2$ and $v_4 = \bar{x}_{n-1}$. If $k_c = \lceil k/2 \rceil$ and $k_f = \lfloor k/2 \rfloor$, then the variables from X and $\bar{X}$ in $v_1, \ldots, v_{k-1}$ are $V_k = \{x_1, \ldots, x_{k_f}\}$, and $\bar{V}_k = \{\bar{x}_n, \ldots, \bar{x}_{n-k_c-2}\}$, respectively.

Now, the k^{th} stage of interpolation involves the following two steps:

1. Interpolating the Dixon polynomial in $k-1$ variables $deg(\delta_{\mathcal{F}}, v_k) + 1$ times and

2. interpolating as many univariate polynomials in v_k as the number of terms in $\delta_{\mathcal{F}}$ in variables $v_1, \ldots, v_{k-1}$.

The complexity of our algorithm depends on the cost of these two steps which in turn depends on the term structure of the Dixon polynomial and also its degree in v_k. In the last chapter, we had derived precise relationships and formulas for the term structure of the Dixon polynomial. We will now use those relationships to determine the complexity of constructing the Dixon matrix using our interpolation-based algorithm.

Complexity

Theorem 5.1.1 *The construction of the Dixon matrix requires* $\mathcal{O}\left(m\, n!^2\, d^{2n}\right)$ *arithmetic operations where* m *is the maximum of the cardinalities of the support of any input polynomial.*

Proof : First we will compute the costs of steps (1) and (2) above, and then we will derive the recurrence relation characterizing the complexity of the algorithm for interpolating the Dixon polynomial using the polytope relationships of the previous chapter. Recall that $\mathcal{F}$ denotes the input polynomial system, X denotes the n variables and $\bar{X}$ the n substitution variables.

Cost of Step (2):

The cost of step (2) in the above algorithm is the product of (2a) the cost of interpolating a single univariate polynomial in v_k and (2b) the number of terms in $\delta_{\mathcal{F}}$ in variables $v_1, \ldots, v_{k-1}$. These two quantities can be determined as follows.

Cost of step 2a: We first determine the cost of (2a), that is, interpolating a single univariate polynomial in v_k. Using *Lagrange interpolation formula* (see Chapter 13 in [45]), interpolating $D+1$ values of a univariate polynomial to produce the polynomial takes $I(D) = \mathcal{O}(D^2)$ time. Since $deg(\delta_{\mathcal{F}}, v_k) = k_c\, d$, only $k_c\, d$ values are required to interpolate the Dixon polynomial as a univariate polynomial in v_k. Therefore, the cost of step (2a) above is $\mathcal{O}(k_c^2\, d^2)$.

Cost of Step 2b: Now, we need to determine (2b), that is the number of terms in $\delta_{\mathcal{F}}$, when viewed purely as a polynomial in variables $v_1, \ldots, v_{k-1}$. The key observation here is that the Newton polytope of an n-degree system Q of type $(\overbrace{1, \ldots, 1}^{n}; d, \ldots, d)$, contains *within* it the Newton polytopes of all other polynomial systems (including $\mathcal{F}$) in n variables whose degree in each variable does not exceed d. From the construction of the previous chapter, it is easily seen as a consequence that the number of terms (in any subset of the n variables) in the Dixon polynomial of Q bounds the number of terms (in the same subset of variables) in the Dixon polynomial of $\mathcal{F}$. By Theorem 4.2.4, the number of terms in the Dixon polynomial of Q in V_k and $\bar{V}_k$ are $\left(k_f!\, d^{k_f}\right)$ and $\left((k_c-1)!\, d^{k_c-1}\right)$, respectively. The product of these two expressions bounds the number of terms (in variables $v_1, \ldots, v_{k-1}$) in the Dixon polynomial of Q, and thus also $\delta_{\mathcal{F}}$. This determines (2b), and hence also the cost of step (2) above.

Cost of Step (1):

Before calculating the cost of step (1), we will need to know the cost of computing a single value of the Dixon polynomial from the cancellation matrix. We denote this cost by B.

Since we know the highest degree of the input polynomial in each variable, before attempting to calculate the value of the Dixon polynomial, we calculate and store all required powers of each variable in a table. If the highest power of a variable, x_i in any input polynomial is d, then the process of calculating and storing all its powers from $1, x_i, \ldots, x_i^d$ in the table takes arithmetic operations proportional to d. Since there are only n variables, our whole table can be constructed in $\mathcal{O}(n\, d)$ arithmetic operations.

Now, each entry of the cancellation matrix is a input polynomial with m terms. Since each term is a product of one coefficient and at most n powers (one corresponding to each variable), its value can be computed in $n+1$ multiplications by looking up the powers from our precomputed table. Therefore, each entry of the cancellation matrix can be evaluated in $\mathcal{O}(m\, n)$ arithmetic operations. Since the cancellation matrix is $(n+1)\times(n+1)$, all its $(n+1)^2$ entries can be evaluated in $\mathcal{O}(m\, n^3)$ arithmetic operations.

Finally, computing one value of the Dixon polynomial also requires computing the determinant of the cancellation matrix, which requires an additional $\mathcal{O}\left(n^3\right)$ arithmetic operations. Adding up the cost of all the subroutines above, a single value of the Dixon polynomial can be evaluated in $B = \mathcal{O}\left(m\, n^3\right)$ arithmetic operations.

Next we determine the cost of step (1) and set up the recurrence for our algorithm. In step (1), the Dixon polynomial is interpolated in $k-1$ variables as many times as $deg(\delta_{\mathcal{F}}, v_k) + 1$. In the 1^{st} stage the values of the Dixon polynomial need to be computed directly by evaluating the determinant of the cancellation matrix, but for the k^{th} stage ($k > 1$), step (1) involves simply interpolating the Dixon polynomial in $k-1$ variables $k_c\, d$ times. If N_k denotes the total cost of the first k stages, then we get the following recurrence relation for $2n$ stages.

$$\begin{aligned} N_1 &= d\,B + I(d), \\ N_k &= k_c\, d\, N_{k-1} + \left(k_f!\, d^{k_f}\right) \left((k_c - 1)!\, d^{k_c - 1}\right) I\left(k_c d\right). \end{aligned}$$

The first term is the cost of step (1) in the algorithm, and the second term is the cost of step (2). It can be easily verified that the solution to this recurrence is

$$N_{2n} = n!^2 d^{2n} \left(B + \frac{2}{d} \sum_{i=1}^{n} \frac{I\left(id\right)}{i}\right) = \mathcal{O}\left(n!^2 d^{2n} \left(m\, n^3 + \frac{2}{d} \sum_{i=1}^{n} \frac{i^2 d^2}{i}\right)\right) = \mathcal{O}\left(m\, n!^2 d^{2n}\right).$$

$\square$

In the previous theorem, the complexity was related to m, the number of terms with non-zero coefficients in the polynomials of $\mathcal{F}$. This is the first time we have introduced such a quantity and it should really be viewed as another measure of sparsity in the input polynomials. In the case of multi-homogeneous polynomials, the complexity of constructing the Dixon matrix can be related to both m and the volume of the input Newton polytopes, as shown in the following corollary:

Corollary 5.1.2 *The construction of the Dixon matrix of a multi-homogeneous system requires $\mathcal{O}\left(m\, l!^{2r}\, mvol\left(\mathcal{F}\right)^2\right)$ arithmetic operations where m is the maximum of the cardinalities of the support of any input polynomial.*

Proof: We already know from the previous chapter that $mvol\left(\mathcal{F}\right) = \mathcal{O}\left(\frac{n!d^n}{l!^r}\right)$. Therefore, the time taken to construct the Dixon matrix is:

$$\mathcal{O}\left(m\, n!^2 d^{2n}\right) = \mathcal{O}\left(m\, l!^{2r}\left(\frac{n!d^n}{l!^r}\right)^2\right) = \mathcal{O}\left(m\, l!^{2r}\, mvol\left(\mathcal{F}\right)^2\right)$$

□

Therefore, we find that the cost of constructing the Dixon matrix of a multi-homogeneous systems involves both the mixed volume and the number of terms with non-zero coefficients in the polynomials (i.e., the cardinality of their support). This is in contrast to the size of its Dixon matrix itself, which depends only on the mixed volume, not on the cardinality of the support.

5.2 Complexity of Resultant Computation

In this section, we discuss the ramifications of the results in the previous chapters and sections on the total complexity of computing the resultant. The process of computing the resultant using our extended Dixon formulation involves two steps:

1. constructing the Dixon matrix, and

2. computing the determinant of the Dixon matrix.

Once the resultant matrix has been set up, its determinant (or the determinant of one of its rank submatrices) gives the resultant or a nontrivial projection operator. We then get the complexity of computing the resultant using the Dixon formulation to be:

Theorem 5.2.1 *Let $\mathcal{F}$ be a multi-homogeneous polynomial set in n variables with $n+1$ polynomials, where r is the number of variable partitions, the number of variables in each partition are $\mathcal{O}(l)$ and the degree of $\mathcal{F}$ in each partition is $\mathcal{O}(d)$. Then, the bit complexity of computing the resultant of $\mathcal{F}$ using our extended Dixon formulation is*

$$\mathcal{O}\left(m\, l!^{2r}\, mvol\,(\mathcal{F})^2 + \left(\frac{r!^l\, l^n}{(n+1)!}\right)^3\ mvol\,(\mathcal{F})^3\right).$$

Proof : The determinant of a Dixon matrix of size S_d can be computed in $\mathcal{O}\,(S_d)^3$ arithmetic operations. Using the expression for the size of the Dixon matrix from

Corollary 4.2.5 and adding to it the complexity of constructing the Dixon matrix gives the total complexity of computing the resultant using the Dixon formulation.□

Similar results have been derived for other formulations also. In the next section, we compare our results for the Dixon formulation with the complexity of computing the resultant using the sparse resultant formulation.

5.3 Comparison with Sparse Resultant Formulation

5.3.1 Construction of the Resultant Matrix

The following theorem [[14], Theorem 3.1.27] gives the cost of constructing the sparse resultant matrix using the subdivision algorithm.

Theorem 5.3.1 *The complexity of constructing the sparse resultant matrix of a multi-homogeneous system $\mathcal{F}$ is $\mathcal{O}\left(e^n\,(m\,n)^{5.5}\ \mathit{mvol}\,(\mathcal{F})\right)$, where m is the maximum of the cardinalities of the supports of polynomials in $\mathcal{F}$.*

So we find another similarity between the extended Dixon and the sparse resultant formulation, namely, that in both cases, the cost of constructing the respective resultant matrices involves both the mixed volume as well as the cardinality of the support of the input polynomials. This is in contrast to the sizes of the resultant matrices themselves, which, as was shown in the previous chapter, depend only on the volumes of the input Newton polytopes and are unaffected by the cardinality of the supports (Theorem 4.3.7).

5.3.2 Complexity of Resultant Computation

Since the total cost of resultant computation also involves evaluating the determinant of the resultant matrix, which is cube of the size of the matrix, we get the following corollary from Theorem 5.3.1:

Corollary 5.3.2 *The complexity of computing the resultant of $\mathcal{F}$ using the sparse resultant formulation is*

$$\mathbf{O}\left(e^n\,(m\,n)^{5.5}\;mvol\,(\mathcal{F}) + e^{3n+3}\,(mvol\,(\mathcal{F}))^3\right).$$

This complexity formula is a quite involved and complicated. For a meaningful comparison of the complexities of computing the resultant using the Dixon and sparse resultant formulations, some simplifying assumptions can be of great assistance. We identify the following four different kinds of simplifying assumptions:

1. The polynomial system is *hollow*, that is, only the monomials on the boundary of the Newton polytope have non-zero coefficients.

2. The polynomial system is *full*, that is, all monomials inside and on the Newton polytope have non-zero coefficients.

3. Multi-homogeneous polynomials with a large number of partitions, that is, when value of r approaches that of n (i.e., $r \rightarrow n$).

4. Multi-homogeneous polynomials with a large number of variables in each partitions, that is, $l \rightarrow n$.

In the following comparison, we consider various combinations of the above four assumptions.

1. **Hollow:** If the polynomial system is *hollow*, then mathematically, it translates to $m = \mathcal{O}(1)$. By substituting this in Corollary 5.3.2, it can be easily shown that the complexity of computing the resultant of hollow polynomial systems using the sparse resultant formulation is $\mathcal{O}\left(e^{3n} mvol(\mathcal{F})^3\right)$. This is irrespective of any additional simplifying assumptions on the number of partitions or the number of variables in each partition.

   Dixon's method, on the other hand, is sensitive to these additional assumptions. Besides being hollow, if $r \to n$ in the given system, then since $l \to 1$, the complexity of computing the resultant using our extended Dixon formulation can be derived from Theorem 5.2.1 as follows:

$$\begin{aligned}
& \mathcal{O}\left(m\, l!^{2r}\, mvol(\mathcal{F})^2 + \left(\frac{r!^l\, l^n}{(n+1)!}\right)^3 mvol(\mathcal{F})^3\right) \\
= \; & \mathcal{O}\left(mvol(\mathcal{F})^2 + \left(\frac{n!}{(n+1)!}\right)^3 mvol(\mathcal{F})^3\right) \\
= \; & \mathcal{O}\left(mvol(\mathcal{F})^2 + mvol(\mathcal{F})^3\right) = \mathcal{O}\left(mvol(\mathcal{F})^3\right).
\end{aligned}$$

   that is, the complexity using the Dixon formulation reduces to $\mathcal{O}\left(mvol(\mathcal{F})^3\right)$. This shows that the Dixon method is cheaper for computing the resultant, in this case, by a factor of e^{3n}.

On the other hand, if $l \rightarrow n$, then it can be shown that the cost of the Dixon method rises to $\mathcal{O}\left(\left(\frac{n!^2}{d^n} + e^{3n}\right) mvol\left(\mathcal{F}\right)^3\right)$. However, assuming a fixed number of variables, this is still no more expensive than the cost of the sparse resultant formulation.

So for hollow systems, even though in the worst case the cost of the Dixon and the sparse formulations is almost the same, as r becomes large and tends to n, cost of the Dixon method reduces substantially, while the sparse resultant method still remains expensive.

2. **Full:** A full polynomial system is one in which all monomials inside the Newton polytopes have a non-zero coefficient in the polynomials. This means that $m = \mathcal{O}\left(d^n/(l!)^r\right) = \mathcal{O}\left(vol\left(\mathcal{F}\right)\right) = \mathcal{O}\left(mvol\left(\mathcal{F}\right)/n!\right)$. We now consider two additional simplifying assumptions:

   (a) $r \rightarrow n$: This means that the number of partitions are large, each containing a small number of variables $l \rightarrow 1$. Along the same lines as for hollow systems, it can be shown that for full systems also, the complexity of computing the resultant using our extended Dixon method in this case is $\mathcal{O}\left(mvol\left(\mathcal{F}\right)^3\right)$. On the other hand, the complexity of the sparse formulation is derived to be:

$$
\begin{aligned}
& \mathcal{O}\left(e^n \left(m\, n\right)^{5.5}\ mvol\left(\mathcal{F}\right) + e^{3n+3}\left(mvol\left(\mathcal{F}\right)\right)^3\right) \\
= \quad & \mathcal{O}\left(e^n \left(n \frac{mvol\left(\mathcal{F}\right)}{n!}\right)^{5.5}\ mvol\left(\mathcal{F}\right) + e^{3n+3}\left(mvol\left(\mathcal{F}\right)\right)^3\right)
\end{aligned}
$$

$$
\begin{aligned}
&= \mathcal{O}\left(\left(\frac{e^n n^{5.5}}{n!^{5.5}} mvol\left(\mathcal{F}\right)^{3.5} + e^{3n+3}\right) mvol\left(\mathcal{F}\right)^3\right) \\
&= \mathcal{O}\left(\left(\frac{e^n n^{5.5}}{n!^{5.5}} \left(\frac{n!d^n}{l!^r}\right)^{3.5} + e^{3n+3}\right) mvol\left(\mathcal{F}\right)^3\right) \\
&= \mathcal{O}\left(\left(\frac{e^{3n} d^{3.5n}}{n^{2n}} + e^{3n}\right) mvol\left(\mathcal{F}\right)^3\right).
\end{aligned}
$$

From this expression, it is clear that, even at best, the sparse formulation will be more expensive than the Dixon formulation at least by a factor of $\mathcal{O}\left(e^{3n}\right)$.

(b) $l \rightarrow n$: When the number of variables in each partition (l) is large and the number of partitions (r) themselves are small, the cost of the Dixon formulation rises to $\mathcal{O}\left(n!mvol\left(\mathcal{F}\right)^3\right)$. The cost of the sparse formulation in this case changes to

$$
\mathcal{O}\left(\left(\frac{e^{6.5n} d^{3.5n}}{n^{5.5n}} + e^{3n}\right) mvol\left(\mathcal{F}\right)^3\right).
$$

Now, when additionally, in this case, $d << n$, it is possible for the sparse formulation to take less time than the Dixon formulation. In such a case, the sparse formulation can possibly be faster than the Dixon formulation by a factor of up to $\mathcal{O}\left(n!\right)$. However, when the degree is much larger than the number of variables, that is, $d >> n$, the Dixon formulation will be faster than the sparse formulation. In fact, under these circumstances, it is possible for the Dixon formulation to be faster by a factor of up to $\mathcal{O}\left(mvol\left(\mathcal{F}\right)^{3.5}\right)$.

So we find that under most circumstances, the complexity of computing the resultant using our extended Dixon formulation is much lesser than the complexity of the sparse formulation.

Example

Again, these bounds wouldn't be useful if they do not reflect in real examples with actual implementations. We implemented an algorithm to construct the Dixon matrix in Maple. An implementation to construct the sparse resultant matrix in Maple can be found in [7]. We use these two implementations in the following example to demonstrate the disparity in the construction times of the two resultant matrices. We demonstrate both cases, the *full* case, i.e., when the polynomial has all the monomials inside its Newton polytope, as well as when it is relatively *hollow*. All timings are on a Sparc Station 10.

We start with a multi-homogeneous system of type $(1, 2, 1; 1, 1, 1)$. We had discussed this example towards the end of Chapter 4. This is a hollow system already in the sense that there are no monomials inside the Newton polytope - all monomials lie on the surface. In particular, there are 12 monomials in each polynomial. The n-fold mixed volume of this system is 12. The Dixon matrix was constructed using our implementation in Maple in 2.3 seconds and its size was 12×12. The 230×230 sparse resultant matrix was constructed using the implementation of [7] in Maple and it took more than 2 hours (7266 seconds). The disparity in the sizes is consistent with the theoretical results of chapter 4. As was shown towards the end of Chapter 4, the ratio of the sparse and the Dixon matrix for this example is about 19.5. Also, for this hollow case, the sparse resultant matrix takes a much longer time to construct than the Dixon matrix. This is also consistent with the theoretical results we have obtained in the previous subsection, where we showed that for hollow systems, Dixon matrix construction can be shown to take smaller time than the sparse matrix construction.

Next, we increase the degree of the first and the third partition in this system. In other words, we now change the system to a multi-homogeneous system of type $(1, 2, 1; 2, 1, 2)$. So the n-fold mixed volume increases to 48. This case will demonstrate how the construction time increases as the mixed volume is increased while keeping the dimension and the partition structure of the system constant.

First we consider the *full* case when all the monomials inside the Newton polytope are present in the polynomials with non-zero coefficients. In particular, there are 27 monomials in each polynomial. The Dixon matrix was constructed in Maple within 316 seconds. The size of the Dixon matrix was 48×48. The sparse resultant matrix was constructed in Maple and its construction took over 39 hours (142907 seconds) in Maple. The size of the sparse resultant matrix was 936×936. So again, we see that both the matrix size as well as the construction time is considerably larger for the sparse resultant matrix than for the Dixon matrix. It should also be pointed out that despite increasing the degree of the multi-homogeneous system, the ratio of sizes is still $936/48 = 19.5$, which is close to ratio for this example calculated theoretically towards the end of Chapter 4. The reason that increasing the degree did not change the ratio is because the ratio depends only on the number of variables (see Theorem 4.3.4) which we did not change.

Now we make the coefficients of all the monomials which don't lie on the surface of the Newton polytope zero. This is the *hollow* situation we talked about above, that is when the number of monomials is much smaller than the number of integer lattice points lying in the Newton polytope. In particular, now the number of monomials in each polynomial decreases from 27 to 12. The mixed volume obviously remains the same (i.e. 48) since we left all the surface monomials intact and the Newton polytope did not change. The Dixon matrix is also still 48×48. Due to a greedy heuristic that is used in the implementation of [7], the size of the sparse resultant matrix also reduces to 230×230, even though the bound is higher than this. However due to a reduced number of terms, the Dixon matrix took only 9 seconds to construct, as opposed to the 407 seconds when all the terms were present. The construction time of the sparse resultant matrix also decreased to about 5692.32 seconds, but is still much larger than the construction time of the Dixon matrix. Moreover, the Dixon matrix here was relatively sparse as opposed to the case when all the terms were present. $\square$

This example highlighted the theoretical results of this and the previous chapter. In the next chapter, we will describe our implementation of the proposed resultant method and use it to solve numerous problems arising in real-world applications. We will demonstrate that as in theory, indeed in real-life too, our extension of the Dixon formulation is a more efficient method for computing the resultant than other existing formulations.

Chapter 6

Experimental Evaluation of the Dixon Formulation

In this chapter, we present empirical evidence which supports the theoretical results of the previous chapters that our extension to the Dixon resultant method is superior to other existing resultant formulations. We demonstrate the practical applicability of our extension of the Dixon formulation by using it to solve numerous real problems. In the first section we describe an implementation of our method which uses efficient polynomial arithmetic techniques to efficiently construct the Dixon matrix and compute a non-trivial multiple of the resultant. Recall that in the last chapter, we had used the polynomial interpolation technique for efficient construction of the resultant matrix. This section will describe yet another application of the polynomial interpolation technique, this time in the context of extracting the resultant from the resultant matrix. We will outline some heuristics which can be used in conjunction with the interpolation technique to develop an efficient procedure for extracting a projection operator from the resultant matrices.

In the second section we present the results of applying our implementation to numerous elimination problems which arise in real-world applications and compare this to the performance of similar implementations of the sparse and Macaulay

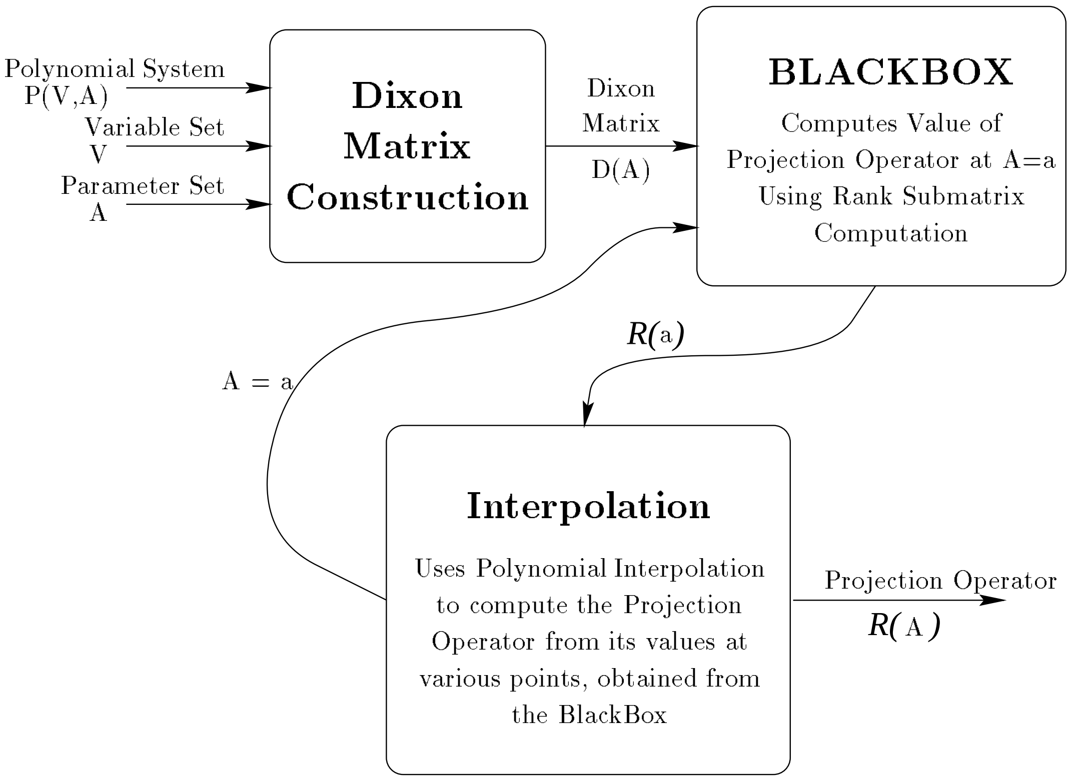

Figure 6.1: Computing a Projection Operator using the Extended Dixon Formulation

formulations to the same problem set. Our empirical comparison demonstrates and confirms the practical superiority of the extended Dixon method for computing resultants.

6.1 An Efficient Implementation

There are two phases of computing a projection operator using Dixon's method. A diagrammatic description of the procedure is given in Figure 6.1. The first is to construct the Dixon matrix, and the second is to extract a projection operator from it.

6.1.1 Construction of the Dixon Matrix

In Chapter 5, we discussed the algorithm which exploits the structure of the input polytopes by using interpolation to construct the Dixon matrix. This algorithm was shown to have complexity which depends on the volumes of the input polytopes. We also demonstrated that the asymptotic complexity of this algorithm is less than the asymptotic complexity of the algorithm for constructing the sparse resultant matrix. However, even though all problems considered in this chapter are from the real world, they do not approach the asymptotic case. For these problems, we found that the Dixon matrix construction algorithm in the classical Dixon formulation performs quite well, and is much easier to implement. In fact, as we will show, even a simple implementation of this classical construction algorithm constructs Dixon matrices must faster than efficient implementations of modern algorithms which construct the sparse resultant matrix. Moreover, as we will show, since in any case the construction time is much smaller in relation to the time taken to extract a projection operator from the resultant matrices, we decided to use the classical approach to construct the Dixon matrix.

The algorithm which we use for constructing the Dixon matrix was outlined in Chapter 2. We implemented that algorithm as a MAPLE program to construct the Dixon matrices. Chapter 3 outlined the algorithm to extract a projection operator from the matrix. The first step in that algorithm is to test if there exists at least one column in the Dixon matrix which is linearly independent of all other columns. Since the entries of the Dixon matrix can be polynomials in the parameters, it can become expensive to perform this test symbolically. Instead, we checked the linear independence of any column in the Dixon matrix probabilistically, by substituting random numbers for parameters and performing the check on the resulting numeric (rather than symbolic) Dixon matrices. The probability of getting a wrong answer using this method is quite low. Moreover, the probability of its success can be arbitrarily increased by performing the check more than once. We have found this method to be very efficient in practice, and it has never given a wrong answer in any of the problems we tried. To make the probability of a wrong answer negligible, we performed this probabilistic test at least three times.

6.1.2 Extracting the Resultant from the Dixon Matrix

Once the test of linear independence of a column in the Dixon matrix is successful, the next step is to extract a projection operator from the Dixon matrix. As was mentioned in chapter 3, the determinant of any non-singular rank submatrix of the Dixon matrix contains the resultant as a factor. So, the objective now is to compute a rank subdeterminant of the Dixon matrix.

Each entry of the Dixon matrix is a polynomial in the parameters of the input system. Since the Dixon matrices are symbolic, using direct Gaussian elimination to compute their rank subdeterminants can lead to large intermediate expressions, and consequently, inefficiency. Instead, we compute their determinants (or rank subdeterminants) using polynomial interpolation. Interpolation techniques can construct multivariate polynomials from their values at various points. The rank subdeterminant was interpolated using our C^{++} implementation of Zippel's sparse polynomial interpolation algorithm [45].

The reader should be cautioned against confusing this phase of interpolation with the interpolation used in chapter 5 to construct the Dixon matrix. Instead, here interpolation is being used to extract a non-trivial multiple of the resultant *from* a symbolic Dixon matrix which has already been constructed. The Dixon matrix could be constructed using any other implementation, independently. This phase takes that Dixon matrix and interpolates a projection operator from it, *without ever* symbolically expanding the resultant matrix. The details of this interpolation phase are as follows:

Given a resultant matrix, a *blackbox* is the function which returns the value of the polynomial (determinant or the rank subdeterminant) for a particular value assignment to the parameters. The blackbox was constructed by

1. evaluating each entry of the symbolic resultant matrix after substituting parameter values, then

2. performing Gaussian elimination on the matrix (which now has integer entries) and finally

3. returning the product of nonzero pivots.

The interpolation algorithm then loops through m stages (where m is the number of parameters), each time lifting one parameter by making a number of calls to the *blackbox*. At the end of the m^{th} stage, the polynomial, which in our case is a nontrivial multiple of the resultant, has been completely interpolated. See [45] for details.

Computing Exact Coefficients

We performed all computations in a finite field modulo a large prime. This means that the projection operator we interpolated is equivalent to the actual projection operator modulo this large prime. Of course, if all coefficients in the actual projection operator are smaller than this prime, then the interpolated projection operator equals the actual one. Otherwise, if the exact projection operator is desired, then there are two alternatives. If a bound on the coefficients of the actual projection operator is known or can be determined, then the computations can be performed in various finite fields modulo different primes until the product of the primes is larger than this predetermined coefficient bound. Finally, the exact coefficients can be lifted from their images in various finite fields using the Chinese remainder theorem. If the coefficients of the projection operator cannot be bounded or if the bound is too loose (and large), then the following approach, which was suggested and used in [45, 33], can be followed. Lift the coefficients successively, using the Chinese remainder theorem, after each finite field computation. Usually, if the projection operator at any given stage is not the actual projection operator, then this lifting process at that stage will change at least some of the coefficients of the projection operator. On the other hand, if the lifting process does not change any coefficient (ie. the coefficients stabilize), then it is quite likely that the projection operator which we have is the actual one. So, in the absence of coefficient bounds, this process can be carried on until the coefficients of the projection operator stabilize. At that point, there is a very high probability that the projection operator we have obtained is the actual projection operator.

Extraneous Factors

Besides the resultant, projection operators typically have some extraneous factors also. That is why, even though the determinants of each rank submatrices of the resultant matrix are different polynomials, they are all valid projection operators. They all have the resultant as a factor, but may have different extraneous factors. In other words, using different rank submatrices of the resultant matrix may introduce a different set of extraneous factors, resulting in different projection operators. As a result, it is important to exercise caution while implementing our method to ensure that all images of the projection operator obtained from the blackbox are consistent. This is ensured by recording and consistently using the same rank subdeterminant to compute all the images of the projection operator.

Since resultant is really the quantity of interest, it is desirable to eliminate, or at least minimize the number of, extraneous factors. One way to do this is by interpolating the greatest common divisor of the determinants of all rank submatrices of the resultant matrix. However, even though this might eliminate some extraneous factors, there is no guarantee that it will eliminate all extraneous factors. Moreover, such a procedure also adds the overhead cost of computing additional determinants (corresponding to each rank submatrix), which increase exponentially as the difference between the rank and size of the resultant matrix increases. Due to the extra cost, we did not use such a method in our problems. Instead, we observed that the extraneous factors are typically powers of very small polynomials, ie. polynomials with a few terms, as compared to the resultant itself. Additionally, if a factor is small, it is usually easy to determine if it is extraneous by substituting some of its roots into the original polynomial system and checking if there it has a zero after this substitution. For instance, given a polynomial system P with coefficients which are parameterized by a and b, one might interpolate a projection operator which has, say $(a-b)^{20}$ as a factor. Then, if substituting b for a in P results in a polynomial system which has no common roots, then we know that $(a-b)^{20}$ is an extraneous factor. This is the heuristic we used in our examples. We interpolated a single projection projection operator, factorized it, and then removed the extraneous factors by visual inspection, and/or back-substitution. In our experience, we found this procedure to be quite inexpensive and effective.

Finally, by profiling the calls to various functions during different interpolation runs, we found that the most expensive operation is *blackbox* computation. Since the blackbox procedure itself involves simply Gaussian elimination, there is not much scope for improvement there. However, the possibilities for improving interpolation lie in reducing the number of calls to blackbox. Two main heuristics we used to reduce the number of calls were as follows:

Determining Tighter Variable Degree Bounds

Zippel's interpolation algorithm assumes the same degree bound (say D) for each parameter at any stage. This assumption is costly for later stages if the actual degree of the parameter being lifted is less than D. The reason is that at the i^{th} stage, the multivariate polynomial in the first $i-1$ parameters has to be interpolated as many times as the degree bound of the i^{th} parameter. If this bound is loose, or too large, then unnecessary extra calls will be made to interpolate the multivariate polynomial in the first $i-1$ parameters.

Fortunately, when the first parameter is being lifted, a loose bound leads to extra calls only to the blackbox – avoiding any actual interpolations. In other words, a loose bound of D is not too costly in the first stage, but becomes successively expensive as the stages progress. Based on this observation, we use the initial crude bound of D to interpolate univariate polynomials in each parameter *separately*, *before* starting the multivariate interpolation. There is very high probability that the degree of, for example, the i^{th} parameter in the univariate polynomial corresponding to this parameter is the same as the degree of the original multivariate polynomial in this parameter. Therefore from these univariate polynomials in each parameter, we determine sharper individual degree bounds on the degree of the multivariate polynomial in each parameter. Later, when interpolating the multivariate polynomial, we use these tighter individual degree bounds at each stage instead of the earlier loose bounds. This results in a speedup of the algorithm since at the k^{th} stage, it reduces the number of times the polynomial in the previous $k-1$ parameters needs to be interpolated, thus reducing the number of calls to *blackbox*.

Order of Interpolation

Once the individual degree bound for each parameter is known, it is conceivable that the order in which the variables are interpolated affects the number of calls to blackbox, and hence the efficiency of interpolation. It can be shown that in the worst case, that is when the number of terms is the maximum possible for a given set of degrees of individual variables, then the order of interpolation does not affect the number of calls to blackbox, or the total time taken for interpolation. For example, if a polynomial in three variables x_1, x_2 and x_3 has degrees $2, 5$ and 6 respectively, then the order of interpolation does not affect the time taken *if* the number of terms in this polynomial is 126. However, if the number of terms is less than this number, then the order of interpolation *may* affect the time taken. The optimal ordering in such a case will depend on the term structure of the polynomial being interpolated. Our execution profiles of the various function calls showed that for our class of polynomials, fewer calls are made to *blackbox* when parameters are lifted in increasing order of their individual degree bounds. It should be pointed out that we do not have a theoretical justification for this observation, and it should only be viewed as a heuristic. This heuristic seemed to work well for the class of problems that we dealt with.

6.1.3 Implementation of Other Formulations

Since both the Macaulay and the sparse resultant formulations involve either determinant or rank subdeterminant computation, our implementation above can be used to extract a projection operator from the resultant matrices in these formulations also. This was accomplished as follows:

For the Macaulay formulation, we developed an implementation in Maple which constructed the Macaulay resultant matrix $M_\sigma(s)$, as well as the denominator matrix $N_\sigma(s)$ (see chapter 2). We used our C^{++} implementation of Zippel's sparse polynomial interpolation algorithm [45] to compute the determinants of these two matrices. Division of the two determinants and extraction of the projection operator from the *GCP* was done on MAPLE.

To construct the sparse resultant matrix, we used Ioannis Emiris' C program based on the algorithm in [15]. A non-trivial multiple of the resultant was extracted from the sparse resultant matrix by rank submatrix computation using our C^{++} implementation of Zippel's sparse interpolation.

In the next section we describe the results of applying these three procedures on various elimination problems arising in different application domains.

6.2 Empirical Results

All empirical results of applying the implementations described in the previous section on various problems are listed together in Table 6.2. The 14 examples used in Table 6.2 are described in the next chapter. All timings are in seconds on a 64 megabyte SUN Sparc10. A (*) in a *Time* column either means that the resultant cannot be computed even after running for more than a day or the program ran out of memory. An N/R in the GCP column means there exists a polynomial ordering for which the Macaulay and the denominator matrix are nonsingular and therefore GCP computation is not required.

Ex	Application	No. of Polys	**Dixon**		
			Matrix Size	Cnstr. Time	*RSC* Time
1	Geom. Reason.	3	13	0.18	342
2	Formula Deriv.	5	14	0.36	76
3	Formula Deriv.	3	5	0.08	*[1]
4	Geometry	3	4	0.08	1021[7]
5	Random	4	6	0.06	*[7]
6	Implicitization	3	10	0.1	0.58
7	Implicitization	3	18	0.45	47
8	Implicitization	3	18	9.33	126
9	Random	4	7	0.21	220
10	Random	5	36	1.1	8130
11	Implicitization	3	11	0.06	3.01
12	Equilibrium	4	5	0.05	0.24
13	Vision	6	6	3.3	0.05
14	Comput. Bio.	3	8	0.067	0.27

Ex	**Macaulay**			**Sparse**		
	Matrix Size	*GCP* Time	*RSC* Time	Matrix Size	Cnstr. Time	*RSC* Time
1	55	*	4608	34	0.3	566
2	330	*	*	86	9.8	4094
3	15	N/R	*[7]	14	0.2	*[7]
4	15	*	*[7]	14	0.2	*[7]
5	84	N/R	*[7]	27	0.7	*[7]
6	45	3741	13.15	16	0.2	0.71
7	66	N/R	604	55	2	519
8	153	*	94697	54	1.9	1369
9	56	*	9972	24	1.2	412
10	1365	*	*	151	40	201558
11	36	N/R	32.58	33	0.7	31.04
12	20	N/R	1.53	20	0.7	2.36
13	792	*	*	60	248	10.53
14	28	11.18	0.68	16	0.2	0.33

Table 6.1: Timings – Resultant Interpolated from Matrices.

Ex	**Dix/** *RSC*	**Mac/** *GCP*	**Mac/** *RSC*	**Sparse/** *RSC*
3	6.56	N/R	26.5	76.2
4	1.6	*	84	290
5	0.9	N/R	850	23.4

Table 6.2: Timings - Direct Determinant Expansion in MAPLE.

Examples 3, 4 and 5 consist of generic polynomials with numerous parameters and dense resultants. Interpolation methods are not appropriate for such examples and timings using straightforward determinant expansion in MAPLE are in Table 6.2.

6.2.1 Discussion and Comparison

In all four methods, the time taken to compute the projection operator depends on the cost of the two steps - (i) constructing the resultant matrix and (ii) interpolating a non-trivial multiple of the resultant from the resultant matrix.

Cost of Resultant Matrix Construction

Cost of resultant matrix construction can be significant for large examples, but as for such examples, the cost of computing a projection operator from the resultant matrix would be even more costly. As was shown in our theoretical results of chapter 6, the cost of resultant computation is dominated by the cost of extracting a multiple of the resultant from the resultant matrix. So, as is expected, even the empirical results of this chapter confirm that the cost of matrix construction is quite insignificant (see the columns labeled *Cnstr. Time* in Table 6.2) for all formulations on all examples.

Nevertheless, we find that Dixon matrices for most problems took much smaller time to construct. It should be pointed out that the construction of Dixon matrices was done on MAPLE using a straightforward implementation without any improvements, whereas the sparse resultant matrices were constructed using an efficient implementation by Ioannis Emiris of his algorithm in C. Despite this, the construction of the Dixon matrix took an insignificant amount of time as compared to the construction of the sparse resultant matrix. For example, the Dixon matrix of the Camera Motion from Point Matches problem (number 13) took only 3.3 seconds using the naive MAPLE construction program on a SPARC 10, whereas the sparse resultant matrix of the same problem took 248 seconds using an efficient C implementation. We speculate that the use of an efficient implementation for constructing Dixon matrices, such as the one used for constructing the sparse resultant matrices, would further reduce the already insignificant Dixon matrix construction time.

Since the total time to compute a projection operator is dominated by the last phase which extracts it from the resultant matrices using polynomial interpolation, we discuss that phase next.

Cost of Resultant Matrix Expansion

As was mentioned earlier, interpolation cost depends on the cost of an individual *blackbox* call and the total number of calls to *blackbox*. The cost of an individual call to *blackbox* includes the cost of substituting values for parameters in the matrix and the cost of performing Gaussian elimination on the resulting matrix. The cost of evaluating an individual entry in Dixon matrix is higher than the cost of evaluating an individual entry in Macaulay, or the sparse resultant matrix because the entries of the Dixon matrix are polynomials of degree at most $n+1$ in the coefficients of the original polynomials whereas the entries in the Macaulay and the sparse resultant matrices are simply the coefficients of the input polynomials. However, the number of such evaluations is determined by the matrix size and is much smaller for the Dixon matrix than for Macaulay and sparse matrices. Consequently, the time taken for substituting values for parameters in the Dixon matrix is smaller than the time taken for substitution in the Macaulay and sparse matrices.

The time taken by a call to *blackbox* is however totally dominated by Gaussian elimination after substitutions have been made into a matrix. Below we compare the four methods based on time taken to perform Gaussian elimination in a *blackbox* call and number of calls to the *blackbox*.

GCP vs. RSC

1. **Cost of Gaussian Elimination:** Size of the resultant matrix of a polynomial system after perturbation is the same as that before perturbation (see Chapter 3). So, the cost of Gaussian elimination is the same in *GCP* and *RSC* methods.

2. **Number of calls to *blackbox*:** Due to the introduction of a new perturbation variable in *GCP* methods, one more iteration (or stage) is needed during interpolation for that variable. This results in more calls to all the procedures including *blackbox* while computing the projection operator using the *GCP* method. Secondly, the introduction of the new perturbation variable can also potentially raise the number of terms in the projection operator, and hence the number of terms in the polynomials at each stage of interpolation. This will raise the number of calls to *blackbox* even further. It may be possible to avoid computing the *GCP* and directly compute its trailing coefficient, but we speculate that this is likely to require as many *blackbox* calls as directly interpolating the whole *GCP*.

In practice, this growth in the number of calls to *blackbox* makes the *GCP* method almost impractical. On the other hand, cost of the *RSC* method remains the same. These observations about the *GCP* method are confirmed by comparing the entries of column labeled Macaulay/*GCP* against the columns Dixon/*RSC*, Macaulay/*RSC* and Sparse/*RSC*.

While working with Macaulay formulation, it is often (9 out of the 14 examples in Table 6.2) the case that one of the two methods, *RSC* or *GCP*, needs to be resorted to, and analysis suggests that *RSC* should be chosen due to its efficiency.

We should point out that the *GCP* method does have a distinct merit in that it always succeeds in extracting the proper affine components of a variety, whereas other methods may fail. However, the failure of *RSC* methods seems to occur very rarely and they have been successfully applied to all nontrivial examples in this thesis.

Dixon vs. Macaulay vs. Sparse

1. **Cost of Gaussian Elimination:** Once the substitution phase is over, the matrix entries are only entries from some finite field, and the cost of Gaussian elimination depends solely on the size of the matrices. Due to the much smaller size of the Dixon matrix, time taken for Gaussian elimination of the Dixon matrix is much less than for the other two resultant matrices. Since Gaussian elimination dominates the *blackbox* call, the *cost of an individual blackbox call is the least for Dixon matrix*, followed by the sparse and finally the Macaulay matrix.

2. **Number of calls to** ***blackbox*****:** The number of calls to *blackbox* depends on the structure of the polynomial being interpolated. If there are no extraneous factors, the polynomial being interpolated in all three cases is the same, and so the number of calls to *blackbox* is also the same in all three methods. If extraneous factors are present, then the cost depends on the number of terms in the projection operator. Usually the number of terms in an expanded projection operator is less if it has fewer extraneous factors. However, note that this is not true in general – a simple example being the resultant $\sum_{i=0}^{n-1} a^{n-i}b^{i}$, which has n terms, whereas $a^n - b^n$ is its projection operator with one extraneous factor, but only 2 terms.

Since the interpolation cost depends mostly on the two factors discussed above, *total time taken to interpolate a non-trivial multiple of the resultant in practice is smallest for Dixon/RSC followed by Sparse/RSC and largest for Macaulay/RSC.*

This gradation of efficiency is reflected in Table 6.2. All the examples were completed using the Dixon/*RSC* method. Sparse/*RSC* also solved all the examples, but always took much longer than Dixon/*RSC*. Macaulay/*RSC* could not complete 2 problems and took much longer than Sparse/*RSC* (for most problems) and Dixon/*RSC* (for all problems).

6.2.2 Summary

In this chapter we empirically compared the four methods to compute a projection operator, viz. Dixon/*RSC*, Sparse/*RSC*, Macaulay/*RSC* and Macaulay/*GCP*. The problem set on which the comparison was made consisted of problems arising in real-world applications. Previously in this thesis, it was theoretically shown that the Dixon formulation exploits the sparsity in the input polynomials to construct smaller resultant matrices. This gives a method for computing a projection operator whose complexity is better than that of the sparse or the Macaulay formulation. This chapter provided empirical evidence in support of these theoretical findings. On this set of 14 problems from real application domains, Dixon/RSC was found to be the most efficient and was able to solve all 14 problems, followed by Sparse/RSC which also solved all problems, but took much (up to 50 times) longer. Next was Macaulay/RSC which could not solve 3 problems, and on others, took much longer than both, Sparse/RSC and Dixon/RSC. Macaulay/GCP was almost impractical due to the introduction of an extra perturbation variable, and could solve only 2 problems.

In the next chapter, we detail some specific problems arising in various applications which were efficiently solved using our implementation of our method.

Chapter 7

Applications

The purpose of this chapter is to impress upon the reader the significance of customizing resultant based methods to yield efficient solutions to problems arising in real applications. To demonstrate this, we have taken two problems each from four very different application areas of computer science, namely:

1. Computer Vision,

2. Geometric Reasoning,

3. Robotics, and

4. Algebra.

For some of these problems, our solutions are the first ever solutions, and for others, the most efficient available. For a comparison of the sizes of various resultant matrices in this chapter, see Table 7. Besides describing the solutions, we also try to compare

Application Description	Section	Dixon	Sparse
Invariants	7.1.1	10	36
Camera Motion	7.1.2	6	60
Side Bisector Reln	7.2.1	13	34
Tetrahedron Volume	7.2.1	14	86
Implicitization	7.2.2	18	55
Cyclic Molecules	7.3.1	28	86
Stewart Platform	7.3.2	56	405
Sum of Squares (n=4)	7.4.1	16	48
Cyclic Roots (n=5)	7.4.2	6	26

Table 7.1: Resultant Matrix Sizes for Various Applications

the performance of the various resultant formulations on each of these problems.

7.1 Computer Vision

7.1.1 Geometric Invariant Computation

Geometric invariants play an important role in object recognition as an aid to building model libraries of objects. Useful invariants are often found by extensive experience and they are based on geometric invariant properties studied by algebraists over many years. Given a geometric configuration, there is however a need to systematically generate and search for its invariants. In our paper [28], we gave a complete solution to computing a single invariant for a geometric configuration, if it exists. The algorithm works in three steps:

1. the problem formulation step in which algebraic relations are established between object parameters and image parameters (or equivalently, parameters of two different images) using an imaging transformation,

2. elimination of transformation parameters resulting in an invariant relation between object and image parameters, and

3. lastly, extraction of a single invariant from the algebraic relation.

A complete solution for the third step, called the *separability* problem was given in our paper [28]. In that paper, we show that there is a simple and efficient solution to the separability problem provided the resultant obtained from the second step is explicitly known. The efficiency of our algorithm is thus linked to the efficiency of computing the resultant. We propose the use of our extension of Dixon's method for computing non-trivial multiples of the resultant.

Dealing with Extraneous Factors: The resultant methods will typically give a projection operator, which may contain other extraneous factors besides the invariant relations. However, in this application, it is quite easy to identify extraneous factors from actual invariant relations. In particular, if the image parameters and object parameters are assigned identical numeric values, then the invariant relation must vanish. So if a given factor of the projection operator does not vanish for such an assignment, then it must be extraneous. In other words, to identify extraneous factors, assign random numbers to the object parameters. Assign identical numbers to corresponding image parameters. Under this assignment, all factors of the projection operator which *do not* vanish have to be extraneous. Remove these factors from the projection operator. Repeated apply this technique, until only one factor is remaining in the projection operator. Then that factor is the required invariant relation. This technique usually removes all extraneous factors in the first iteration itself. Even otherwise, it is usually a trivial job to identify the invariant relation and separate out the extraneous factors using this technique.

In this section, we present one of the examples which demonstrates the application of resultant methods to computing invariants:

A quadratic and a line in 1-d under projective transformation: Let the line (with parameters l_1 and l_2) and the quadratic (with parameters c_1, c_2 and c_3) be

$$\begin{aligned} l_1 x + l_2 &= 0 \qquad \text{and} \\ c_1 x^2 + 2c_2 x + c_3 &= 0 \end{aligned}$$

respectively. Let the transformation (with transformation parameters a_1, a_2, b_1 and b_2) be

$$X = \frac{a_1 x + a_2}{b_1 x + b_2}.$$

Let the image parameters after the projective transformation be L_1, L_2, C_1, C_2 and C_3. Then the transformation equations relating the configuration parameters before and after the transformation can be derived to be:

$$C_1 - c_1 b_2^2 + 2c_2 b_2 b_1 - c_3 b_1^2 = 0$$

$$
\begin{aligned}
C_2 - c_2b_2a_1 + c_3b_1a_1 + c_1b_2a_2 - c_2a_2b_1 &= 0 \\
C_3 - c_3a_1^2 - c_1a_2^2 + 2c_2a_2a_1 &= 0 \\
L_1 + l_1b_2 - l_2b_1 &= 0 \\
L_2 - l_1a_2 + l_2a_1 &= 0.
\end{aligned}
$$

The elimination problem is that of computing the resultant of the above five equations by eliminating the four transformation parameters a_1, a_2, b_1 and b_2.

We used the method presented in this thesis to construct its Dixon matrix which was of size 10×10. The size of the sparse resultant matrix for this problem turns out to be much larger, at 36×36. Although both formulations gave rise to extraneous factors, using the technique of substituting random numbers for image and object parameters into the factors, previously described in this section, it was trivial to identify the invariant relation. Since the Dixon matrix is much smaller, it was easier to construct. It also took a much smaller time to interpolate a projection operator from the Dixon matrix than the sparse resultant matrix.

After identifying the invariant relation and separating it into the absolute invariant using our algorithm in [28], we get the invariant to be:

$$
I = \frac{c_2^2 - c_1c_3}{c_1l_2^2 - 2c_2l_2l_1 + c_3l_1^2}.
$$

For other examples and details of our invariant algorithm, the reader is referred to our paper [28]. The method in this thesis has been also been applied to compute thermal invariants of vehicles from their infrared images, and details of that application were presented in our AAAI paper [34].

7.1.2 Determining Camera Motion from Point Matches

This is a well known problem in computer vision where two images of the same scene taken by the same camera from two different positions are given, and the displacement of the camera between the two positions has to be computed.

Let q and d be the 3-dimensional vectors representing the initial and the final positions of the camera. Clearly, camera motion can be easily determined from these two positions. The objective then is to determine q and d, which contain 6 unknowns. The formulation we present here was investigated by Emiris. The problem has 6 polynomials in the 6 variables corresponding to q and d. Five of the variables are eliminated to compute the resultant as a polynomial in the sixth.

Let a_i and a'_i be 3-dimensional vectors representing the known positions of the i^{th} point in the two images. The polynomial system is as follows:

$$\begin{aligned} p_i &= (a_i^T\, q)(d^T\, a'_i) + (a_i^T\, a'_i) + ((a_i \times q)^T\, a'_i) \\ &\quad + ((a_i \times q)^T\, (d \times a'_i)) + (a_i^T\, (d \times a_i)) \qquad \text{for } 1 \leq i \leq 5 \\ p_6 &= 1 - d^T q \end{aligned}$$

Emiris [14] showed that this system can be approximated well as a multihomogeneous system of type $(2, 3; 1, 1)$. Emiris reported that the sparse resultant matrix of this system is 60×60. We applied our implementation of our extension of the

Dixon formulation and demonstrated that this is much larger in comparison to the Dixon matrix of this system, which is only 6×6. Moreover, the resultant computed as the determinant of the Dixon matrix *does not* have any extraneous factors. So not only do we find that the Dixon formulation results in a smaller resultant matrix, but in this case, also gives the exact solution!

7.2 Geometric Reasoning & Modeling

7.2.1 Formula Derivation

Elimination methods such as resultant computation can, very naturally, be used to derive geometric identities. In fact, this was one of the applications which we used to motivate the study of Elimination methods in general, and resultant in particular. In this section, we present two such derivations, both of which appear in the empirical results table of the last chapter. The first formula, relating the sides and bisectors of a triangle, was derived in the first chapter without giving any details. In this section we describe how resultants can be used to efficiently derive this formula. The second formula is we will derive is of the maximum volume of a tetrahedron, given the areas of its four surfaces.

Side-Bisector Relation: Recall from the first chapter that ABC is a triangle as in figure 7.1, a, b and c the length of the sides BC, AC and AB, a_i and a_e the length of internal and external bisectors of angle A, and b_e the length of the external angle bisector of angle B. The objective is to express a in terms of a_i, a_e and b_e. This problem was solved using a computer in [20] and we have presented it exactly in the same way.

As was argued in Chapter 1, the expression for a can derived by eliminating b and c from equations 1.1. This can be achieved by computing the resultant of these

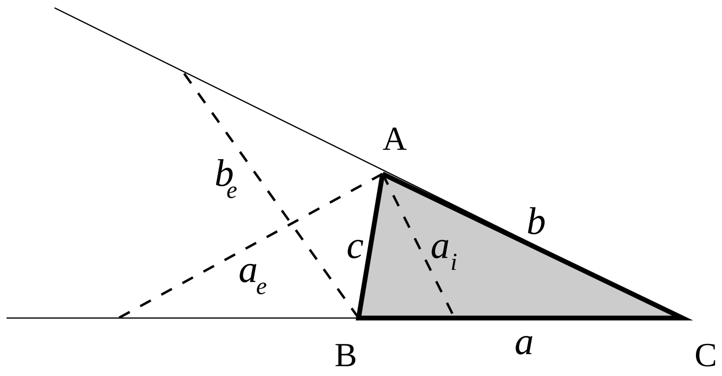

Figure 7.1: Side-Bisector Relation

three polynomials w.r.t. the two variables, b and c. To this effect, let us first represent these equations as polynomials by transforming them to the following:

$$
\begin{aligned}
q_1 &= a_i^2(b+c)^2 - cb(c+b-a)(c+b+a) \\
q_2 &= a_e^2(c-b)^2 - cb(a+b-c)(c-b+a) \\
q_3 &= b_e^2(c-a)^2 - ac(a+b-c)(c+b-a)
\end{aligned}
$$

The Dixon matrix of these polynomials turns out to be 13×14. The total time for constructing the Dixon matrix and performing the column linear independence test probabilistically took less than 0.18 seconds. The total time to interpolate a projection operator from the Dixon matrix using our implementation took 342 seconds. After removing extraneous factors, the smallest necessary condition was obtained and it contains 330 terms. The result is the same as the one reported in [20]. Since the expression is of degree 20 in a, the triangle cannot be constructed using a compass and a ruler, given a_i, a_e and b_e (see corollary 2 of Theorem 5.4.1 in [24]).

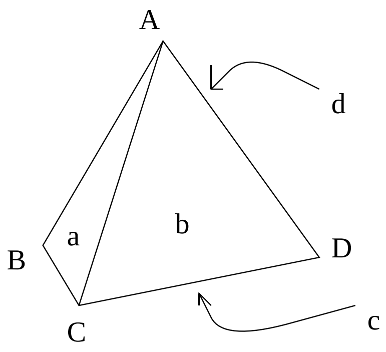

Figure 7.2: Maximum Volume of a Tetrahedron

Gao and Wang in [20] reported that they solved the problem by successively eliminating b and c using a combination of pseudo division and Sylvester's resultant computation. They took about 19 hours on their implementation in lisp on a SUN 4 workstation. Our algorithm took less than 10 minutes on a SUN 4, and less than 6 minutes on a SPARCstation 10. It should also be pointed out that the Sparse resultant matrix for this problem is 34×34, and the Macaulay matrix is 55×55. Not only are these matrices much larger than the Dixon matrix, they also take a much longer time for the derivation, as the Sparse formulation takes 566 seconds, the Macaulay/RSC takes 4608 seconds, while Macaulay/GCP is unable to solve the problem. $\square$

Maximum Volume of a Tetrahedron: Consider a tetrahedron as in figure 7.2. The objective is to determine the maximum volume that a tetrahedron can have, given that the squares of the areas of the four faces, ABC, ACD, BCD and ABD are a, b, c and d respectively.

It has been established in [21] that if there exist parameters x, y, z and w which satisfy the following four equations:

$$
\begin{aligned}
yz + zw + wy - a &= 0 \\
zx + xw + wz - b &= 0 \\
wx + xy + yw - c &= 0 \\
xy + yz + zx - d &= 0
\end{aligned}
$$

then the tetrahedron is an orthocentric tetrahedron and hence the one with the maximum volume for these surface areas. Moreover, the square of the volume (T) of this tetrahedron is:

$$
T = \frac{2}{9}(xyz + yzw + zwx + wxy)
$$

How does one get the value of T purely in terms of a, b, c and d? This problem translates to eliminating x, y, z and w from the above mentioned five equations, i.e., computing the resultant of the following five polynomials:

$$
\begin{aligned}
q_1 &= yz + zw + wy - a \\
q_2 &= zx + xw + wz - b \\
q_3 &= wx + xy + yw - c \\
q_4 &= xy + yz + zx - d \\
q_5 &= 2(xyz + yzw + zwx + wxy) - 9T
\end{aligned}
$$

with respect to x, y, z and w. It turns out that any of these variables x, y, z and w being zero is a degeneracy; hence we can work under the constraint $\mathcal{C} \equiv x \neq 0 \wedge y \neq 0 \wedge z \wedge w \neq 0$.

The Dixon matrix of this set is 14×14 and its column corresponding to the monomial x is linearly independent of the others. The total time for constructing the Dixon matrix and performing the linear independence test took 0.36 seconds. Then the resultant was interpolated from the Dixon matrix, which took 76 seconds. It was found that there was *no extraneous factor* in the necessary condition, hence it is the smallest necessary condition and it contained 434 terms. The total time taken was just above 76 seconds.

On the other hand, the sparse resultant matrix of this problem was 86×86 and it took 566 seconds to interpolate the projection operator from it. Moreover, the projection operator obtained from the sparse formulation had extraneous factors. The Macaulay matrix was 330×330, and the resultant could not be interpolated from it, even using the RSC method. $\square$

7.2.2 Curve & Surface Implicitization

The problem of curve and surface implicitization is a standard problem in solid modeling, for which efficient solutions are being continuously sought. As was mentioned in Chapter 1, one of the benchmark examples in implicitization is the Bicubic surface implicitization, which was proposed as a challenge by Hoffman [26]. When the problem was posed in 1990, the state of the art elimination methods took about 10^5 seconds to implicitize it. We show here how this problem can be efficiently solved using the methods in this thesis.

Let s and t be the variables used to parametrize the surface, and let x, y and z be the rectilinear coordinates of the points on the surface. Recall from Chapter 1 that the following parametric representation of the coordinates of the points on a bicubic surface is given, and the objective is to derive the implicit equation of the surface.

$$\begin{aligned}
x &= 3\,t\,(t-1)^2 + (s-1)^3 + 3\,s \\
y &= 3\,s\,(s-1)^2 + t^3 + 3\,t \\
z &= -3\,s\left(s^2 - 5\,s + 5\right)t^3 - 3\,\left(s^3 + 6\,s^2 - 9\,s + 1\right)t^2 \\
&\quad +t\left(6\,s^3 + 9\,s^2 - 18\,s + 3\right) - 3\,s\,(s-1).
\end{aligned}$$

One way to derive the implicit equation of the bicubic surface is by eliminating the parameters s and t from these three equations, and the implicit equation would then be the resultant, which is a polynomial in x, y and z. This is easily accomplished by computing the resultant of these polynomials with respect to s and t.

We tried to derive the implicit equation using all three resultant formulations. The Macaulay resultant matrix for this problem is of size 66×66, and it took 604 seconds to derive the implicit equation. The sparse resultant matrix was slightly smaller, being 55×55, and this formulation did slightly better by carrying out the implicitization in 519 seconds. Our implementation of our extension to the Dixon formulation performed most efficiently by deriving the implicit equation within 47 seconds, while the size of the Dixon matrix was only 18×18. Moreover, our method was the only one which did not produce any extraneous factors, whereas both sparse and Macaulay formulations gave rise to extraneous factors.

7.3 Robotics & Kinematics

7.3.1 Conformal Analysis of Cyclic Molecules

This is a problem from the emerging field of Computational Biology, and is typically solved by viewing it as a special case of a kinematics problem. This problem was solved using sparse resultant formulation by Emiris [14], and we present his formulation here. We will only present the polynomial system in question, and for details of the problem formulation, see Emiris's thesis [14].

In the following system, t_1, t_2 and t_3 are variables and βs are constants. The objective is to compute the roots of this 3-variable polynomial system:

$$\begin{aligned}
p_1 &= \beta_{11} + \beta_{12}t_2^2 + \beta_{13}t_3^2 + \beta_{14}t_2^2t_3^2 + \beta_{15}t_2t_3 \\
p_2 &= \beta_{21} + \beta_{22}t_3^2 + \beta_{23}t_1^2 + \beta_{24}t_3^2t_1^2 + \beta_{25}t_3t_1 \\
p_3 &= \beta_{31} + \beta_{32}t_1^2 + \beta_{33}t_2^2 + \beta_{34}t_1^2t_2^2 + \beta_{35}t_1t_2
\end{aligned}$$

The first method to compute the roots is by computing their u-resultant. This is done by adding another polynomial:

$$p_4 = u + u_1t_1 + u_2t_2 + u_3t_3$$

to the system and eliminating t_1, t_2 and t_3 from the four polynomials. Emiris [14] reported that this results in a sparse resultant matrix of size 86×86. This is much larger than the Dixon matrix which is only 28×28. Moreover, here again, the resultant computed by our extended Dixon formulation has no extraneous factors.

The second method of computing the roots is by eliminating any two of the three variables from the original three polynomials to compute the resultant in the third variable. Emiris reported that the sparse resultant matrix under this formulation was 16×16. Even here, we find that the Dixon matrix is smaller with a size of only 8×8.

7.3.2 Stewart Platform

The Stewart platform problem is a standard benchmark elimination problem of robotics. The quaternion formulation we present here is by Emiris [14]. It contains 7 polynomials in 7 variables, 6 of which are to be eliminated. Let $x = [x_0, x_1, x_2, x_3]$ and $q = [1, q_1, q_2, q_3]$ be two unknown quaternions, to be determined. Let $q^* = [1, -q_1, -q_2, -q_3]$. Let a_i and b_i, for $i = 2, \ldots, 6$, be known quaternions, and let α_i, for $i = 1, \ldots, 6$, be six predetermined scalars. The 7 polynomials are:

$$
\begin{aligned}
f_1 &= x^T x - \alpha_1 q^T q, \\
f_i &= b_i^T (xq) - a_i^T (qx) - (q b_i q^*)^T a_i - \alpha_i q^T q, \qquad i = 2, \ldots, 6, \\
f_7 &= x^T q^*.
\end{aligned}
$$

Out of the 7 variables, $x_0, x_1, x_2, x_3, q_1, q_2, q_3$, any six are to be eliminated to compute the resultant as a polynomial in the seventh. It is known (see [38]) that this system has exactly 40 complex roots, therefore the resultant is a degree 40 polynomial. We successfully computed the projection operator of this problem using our extension of the Dixon formulation. However, we were unsuccessful in applying the sparse resultant formulation to compute a projection operator. We first report our experiments with the Sparse resultant formulation and then with our extension of the Dixon formulation on this problem.

Emiris reported in his thesis [14] that the sparse resultant matrix of this system is 405×405. We could not find in his thesis the time it took his program to construct this matrix. Moreover, he reported (see section 5.3 in [14]) that the univariate matrix polynomial corresponding to this matrix is identically singular, therefore it seems that he was not able to find the roots, or the resultant of this system using the sparse resultant formulation. Moreover, due to lack of memory space, we have not been able to construct a sparse resultant matrix for this system on a SPARC station 10. Therefore, we were also unsuccessful in using the RSC method to extract a projection operator, or even construct a sparse resultant matrix of the Stewart problem.

Nevertheless, we successfully computed the projection operator of the Stewart problem using our extension of the Dixon formulation. Our experiments have shown that if we eliminate $x_0, \ldots, x_3, q_2, q_3$ from this system, then its Dixon matrix has size only 56×56 and it takes only 94.65 seconds to construct it. Moreover, it took only 71.06 seconds to interpolate a projection operator, which was a polynomial of degree 103 in q_1. So, even though we were able to easily compute a projection operator for this system using our method, eliminating this set of variables resulted in extraneous factors of degree 63 ($= 103 - 40$). But when we eliminated $x_0, \ldots, x_2, q_1, \ldots, q_3$, then the Dixon matrix of this system has size 116×116, it was constructed in 273.5 seconds and it took 256.79 seconds to interpolate a projection operator. However, this projection operator, which is a polynomial in x_3, only has degree 40, which implies that it does not have any extraneous factor. Therefore, even though our method took slightly longer time for this ordering, it dramatically eliminated the number of extraneous factors.

7.4 Algebra

7.4.1 Sum of Squares Problem

Let $\Sigma_n = \{A_1, A_2, \ldots, A_{n+1}\}$ be the following system of $n+1$ polynomials:

$$
\begin{aligned}
A_1 &= x_1^2 - s_1 \\
A_2 &= x_2^2 - s_2 \\
&\vdots \\
A_n &= x_n^2 - s_n \\
A_{n+1} &= y - \sum_{i=1}^{n} x_i.
\end{aligned}
$$

Let all x_i, s_i and y be variables. If we eliminate x_i from these polynomials, then we will get the resultant which is a polynomial in y and the s_i. It turns out that if the resultant is viewed as a polynomial only in y, then its roots are all possible sums of the square roots of s_i. Clearly, since there are only 2^n such sums possible, the degree of resultant in y should be 2^n. Also, if one were constructing a resultant matrix whose entries were simply the coefficients of the A_i, when viewed as polynomials in x_i (in other words, entries are either y or one of the s_i), then the optimal resultant matrix would be of size $2^n \times 2^n$.

It can be shown that the Dixon matrix, D_n, of the above system can be recursively represented as follows:

$$
\begin{aligned}
D_0 &= [y] \\
D_n &= \begin{bmatrix} D_{n-1} & s_n\, I_{2^{n-1}} \\ I_{2^{n-1}} & D_{n-1} \end{bmatrix}, \quad \forall n \geq 1.
\end{aligned}
$$

where I_k is a $k \times k$ identity matrix.

It is easy to see that the Dixon matrix is of size $2^n \times 2^n$, and therefore an optimal resultant matrix of this system. In fact, it can also be proven that the sparse resultant matrix of this system is of size $2^n + n\, 2^{n-1}$ and the Macaulay matrix is even larger, of size $\binom{2\,n+1}{n}$. In fact, all of following can be easily proven.

1. Dixon matrix is the smallest resultant matrix.

2. Dixon matrix is an optimal matrix.

3. There are no extraneous factors in the projection operator computed using the Dixon formulation. In other words, the Dixon formulation computes the exact resultant.

4. There *are* extraneous factors in the projection operator computed using the sparse and Macaulay formulations.

n	degree of resultant	Matrix Size		Constr. Time	
		Dixon	Sparse	Dixon	Sparse
3	6	2	6	0s	0s
4	20	6	25	0s	0s
5	85	22	147	0s	21s
6	290	78	887	7s	5448s

Table 7.2: Empirical Data for Standard Cyclic Roots Problem

Just to demonstrate how this difference in sizes translates into practice, for $n = 4$, the Dixon matrix would be of size 16, the sparse matrix of size 48 and the Macaulay matrix will be of size 126.

In fact, our results concerning optimality of the Dixon resultant matrix hold even when A_i, for $1 \leq i \leq n$, are arbitrary second degree polynomials in x_i. It follows that the Dixon matrix of any other problem which is a subclass of such a system is also optimal. As an example, the problem of determining the binary representation of an integer can easily be expressed as the problem of solving n second degree univariate polynomials and one linear polynomial with n variables. Therefore, since this is a subclass of the general problem described above, its Dixon matrix will also be optimal.

7.4.2 Cyclic Roots Problem

The cyclic n-roots problem is a classic benchmark in algorithmic ideal theory. Computing its Gröbner basis has proven quite difficult in practice. In this section, we concentrate on computing the resultant of this polynomial system. We first discuss the standard formulation of this problem, and then a simplified one by Emiris. Instead of stating the problem for arbitrary n, we just state it for $n = 4$, and the reader should be able to generalize. The emphasis in this section will be to compare

n	degree of resultant	Matrix Size		Constr. Time	
		Dixon	Sparse	Dixon	Sparse
4	6	2	6	0s	0s
5	20	6	26	0s	0s
6	66	18	102	1s	3s
7	270	72	627	55s	1508s

Table 7.3: Empirical Data for Alternative Formulation of Cyclic Roots Problem

the resultant matrix sizes. Since this is a benchmark example of a *mixed* polynomial system, we will highlight the difference in the sizes of the Dixon and the sparse resultant matrix as the dimension of the system rises.

Standard Formulation: Eliminate x_1, x_2, x_3 from

$$\begin{aligned} x_1 + x_2 + x_3 + x_4 &= 0 \\ x_1x_2 + x_2x_3 + x_3x_4 + x_4x_1 &= 0 \\ x_1x_2x_3 + x_2x_3x_4 + x_3x_4x_1 + x_4x_1x_2 &= 0 \\ x_1x_2x_3x_4 - 1 &= 0 \end{aligned}$$

The empirical results for different values of n are in Table 7.2.

Another Formulation: This is an equivalent formulation from Emiris's thesis. It reduces one variable from the problem by dividing the i^{th} equation by x_n^i, and substituting y_i for x_i/x_n. For $n = 4$, this translates to eliminating y_1, y_2 from

$$y_1 + y_2 + y_3 + 1 = 0$$

$$
\begin{aligned}
y_1y_2 + y_2y_3 + y_3 + y_1 &= 0 \\
y_1y_2y_3 + y_2y_3 + y_3y_1 + y_1y_2 &= 0
\end{aligned}
$$

Then later one can solve $y_1y_2y_3 = x_4^{-4}$ to get the solution in terms of the xs. The empirical results for different values of n are in Table 7.3.

So once again we find that in this problem also, the Dixon matrix is the smallest, and it is the easiest to construct.

Chapter 8

Conclusion & Future Directions

8.1 Conclusions from the Thesis

In this thesis, we extended the applicability of the classical Dixon formulation to problems from numerous application domains. We proposed a method which uses linear algebra techniques to extract a non-trivial multiple of the resultant as the determinant of any non-singular rank submatrix of the resultant matrix – degenerate or not. Our method can also be used to extract non-trivial multiples of the resultant from degenerate Macaulay and Sparse resultant matrices. In this light, our method is proposed as an alternative to the technique of Generalized Characteristic Polynomial proposed by Canny [5] for computing resultants when the Macaulay formulation gives an identically zero projection operator. The major advantage of our technique is that it does not add any new terms to the given polynomial system, thus preserving the sparsity and degree of the problem. Our method computes the resultant of polynomial systems with degenerate resultant matrices almost as quickly as the resultant of *well-behaved* polynomial systems. This is in sharp contrast to the GCP technique, which raises the complexity of the problem so much that it renders the technique almost useless for most problems.

Most practical problems arising in various application domains have considerable sparsity. For a resultant method to be useful in real applications, it must exploit this sparsity to compute the resultant efficiently. This thesis showed that our method based on Dixon formulation *implicitly* exploits the sparse structure of the input polynomials, as defined by their Newton polytopes. Based on this observation, it was proved that the size of the Dixon matrix and the complexity of computing the resultant using our method depend on the structure of the Newton polytopes of input polynomials, not their total degree. An exact formula for the size of the Dixon matrix of *unmixed* polynomial systems in terms of their Newton polytopes is derived. This relationship is exploited to tightly bound the size of the Dixon matrices for various classes of polynomial systems such as multi-homogeneous systems. Using these bounds on the size of the Dixon matrix, an efficient algorithm which uses dense polynomial interpolation to construct Dixon matrices is devised. This thesis, for the first time, establishes a definitive link between the classical Dixon resultant formulation and sparsity in input polynomials, as determined by the structure of their Newton polytopes.

Finally, we demonstrated the empirical superiority of our method by developing an efficient implementation and using it to solve real problems from a variety of application domains. Our implementation uses polynomial interpolation to compute the resultant from the Dixon matrix. The superiority of our method is established by solving specific problems from concrete applications such as computer vision, robotics, solid modeling, geometric modeling and curve implicitization. For all the problems we tried, the Dixon matrix was the smallest and resultant computation using our method was the fastest.

8.2 Future Directions

This thesis raises many new questions regarding the Dixon formulation. In this section, we discuss some open theoretical, practical and applications-related questions about our proposed method. Some of these questions are of a more general nature, and can be raised in the context of other resultant methods as well.

1. **Extraneous Factors:** All multivariate resultant methods, including the one proposed in this thesis, compute a non-trivial multiple of the resultant. In other words, they compute the resultant possibly multiplied with some extraneous factors. These extraneous factors are undesirable, and they create problems in certain applications. There is a need to develop techniques to eliminate, or at least reduce, the number of extraneous factors from all the multivariate resultant methods.

   Macaulay had devised a method where for *generic* polynomials, he could exactly identify the extraneous factor as the determinant of a submatrix of the resultant matrix. However, this technique does not work in the non-generic case and for other formulations. It may be possible to identify similar submatrices in the resultant matrices from various other formulations under non-generic conditions.

   One way to reduce extraneous factors is by computing the projection operator using various variable orderings, and then computing their GCD. Alternatively, one can also compute the projection operators corresponding to different rank submatrices of the resultant matrix, and then compute their GCD. This technique sometimes works, but is computationally expensive as the resultant subroutine has to be invoked many times.

Lastly, one can also develop application specific techniques for efficient identification of extraneous factors. In certain applications, the resultant has a specific structure, and it may be possible to make a determination as to whether a factor is extraneous or not depending on whether it has that structure. The technique we used to identify extraneous factors in the invariant computation problem of section 7.1.1 falls in this category. In fact, for that application, identification of extraneous factors could be done very efficiently by random number substitutions, thus avoiding any symbolic computation, or root finding.

2. **Detecting Solvability:** If a given polynomial system has a solution, no matter what the values of its parameters (ie. the system has a solution space of dimension $\geq m$-dimension, where m is the number of parameters), then its resultant is identically zero. Since our method tries to derive a identically non-zero projection operator, which in this case doesn't exist, it will fail. However, our method can fail sometimes, though rarely, even when there exists a identically non-zero resultant. So when failure occurs, there is no way to know whether it occurred because of the former reason or not! As an example of this situation, consider the following example:

   Example (Reason for Failure): Consider the following three bivariate polynomials in which x and y are the variables and a and b are parameters:

$$
\begin{aligned}
p_1 &= ax^2 + bxy + ay + 3 \\
p_2 &= 2axy + ay + b^2
\end{aligned}
$$

$$p_3 \;=\; 4bxy^2 + 4ay^2 - 2b^2x + (12 + a)y + b^2$$

If we eliminate x and y from these three polynomials using our method, then we get a 6×5 Dixon matrix whose rank is 2. However, none of the columns in the matrix is independent of the other columns. So, our condition fails. Now, even though this failure could have occurred because this system has a solution, no matter what the values of the parameters a and b, i.e., solution space of this system is $\geq$ 2-dimensional, we have no way of knowing that. If we had this knowledge, then at least we would know whether the resultant is identically zero, or not (in which case we could look for it using other resultant methods).

It turns out that in this case the solution is indeed 2-dimensional because $p_3 = 4yp_1 + (1 - 2x)p_2$, but there was no way to tell that to be the case given a trace of our procedure on these polynomials. □

There is thus a need for methods to detect if there is a solution to the given system for all values of parameters, in case the method fails to extract a non-zero projection operator.

3. **Dealing with Resultant Matrix Column-Dependence:** Even when we have prior knowledge about the existence of a non-zero resultant, but our method fails, there is no recourse. There is thus a need to develop a technique to extract a non-trivial multiple of the resultant from the Dixon matrix, even when our method fails. Such a technique would render our method complete because the extraction of a non-trivial multiple of the resultant will be guaranteed, whenever it exists.

4. **Bounds for Mixed Systems:** For unmixed systems we were able to completely describe the Dixon matrix in terms of the Newton polytope of the input system. However, for mixed systems, such an analysis has proven very difficult. Future work needs to be done to derive a formula for the size of the Dixon matrix of mixed systems, similar to the one for unmixed systems. At the very least, it may be possible to develop a general scheme for deriving tight bounds on the size of the Dixon matrix for sub-classes of mixed polynomial systems such as the *sum of square roots* problem, polynomial systems where each polynomial is univariate (though in different variables) and the *cyclic n-roots* problem. In the previous chapter, we formally analyzed the first two problems and proved that their Dixon matrix is of optimal size, even though these are mixed systems. For the cyclic n-roots problem, we provided convincing empirical evidence which showed that the Dixon matrix is the smallest resultant matrix and also takes least construction time. However, a formal analysis deriving a formula or bound on the size of the Dixon (or sparse) resultant matrix for this system is also required.

5. **Isolated Solutions of Polynomial Systems:** In his thesis [33], Manocha reduced the problem of finding solutions to 0-dimensional polynomial systems to an eigenvalue problem and used Macaulay matrices to solve for polynomial systems. Later Emiris was able to extended this technique to use sparse resultant matrices to solve polynomial equations. This exploited the advantages of the smaller Sparse resultant matrices to solve polynomial systems even faster than what was possible using Macaulay matrices. Since Dixon matrices are smaller than sparse resultant matrices also, extending this technique so that it reduces polynomial system solving to an eigenvalue problem on the Dixon

matrices, would yield an even more efficient method. However, this extension is not as straightforward as it was to extend from Macaulay to sparse. This is because the structure of sparse and Macaulay matrices is quite similar, in the sense that rows in both consist of input polynomials, multiplied with certain monomials. However, rows of Dixon matrices are more complicated functions of the input polynomial, and further work is required to extend this technique of solving polynomial systems to the Dixon formulation.

Bibliography

[1] Bayer D., Stillman M., *Macaulay User's Manual*, Cornell University, Ithaca, NY.

[2] Becker T., Weispfenning V., in cooperation with Kredel H., *Gröbner Bases, A Computational Approach to Commutative Algebra*, Springer-Verlag, New York, 1993.

[3] Bernshtein D.N., The Number of Roots of a System of Equations, *Funktsional'nyi Analiz i Ego Prilozheniya,* 9(3):1-4, 1975.

[4] Buchberger B., Gröbner bases: An Algorithmic Method in Polynomial Ideal Theory, *Multidimensional Systems Theory,* N.K. Bose, ed., D. Reidel Publ. Co., 1985.

[5] Canny J., Generalized Characteristic Polynomials, *Journal of Symbolic Computation*, 9:241-250, 1990.

[6] Canny J. and Emiris I., An Efficient Algorithm for the Sparse Resultant, *Proc. of AAECC 93,* LNCS 263, Springer-Verlag, May 1993, pp. 89- 104.

[7] Canny J. and Pedersen P., An Algorithm for Newton Resultant, Technical Report, *Cornell University*, CornellCS:TR93-1394, 1993.

[8] Cayley A., On the Theory of Elimination. *Cambridge and Dublin Mathematical Journal,* III, 1865, 210-270.

[9] Chionh E., *Base Points, Resultants and Implicit Representation of Rational Surfaces.* Ph.D. thesis, Dept. Comp. Sci., Univ. of Waterloo, 1990.

[10] Chou S.C, *Mechanical Geometry Theorem Proving*, Reidel, Dordrecht, 1988.

[11] Collins G.E., The Calculation of Multivariate Polynomial Resultants. *JACM,* 18 (4), (1971), 515-532.

[12] Dixon A.L., The Eliminant of Three Quantics in Two Independent Variables. *Proc. London Mathematical Society,* 6:468-478, 1908.

[13] Ehrart E. Sur un problème de géométrie diophantienne, I. Polyèdres et réseaux., *J. Reine Angew. Math.*, 226:1-29, 1967.

[14] Emiris I., *Sparse Elimination and Applications in Kinematics*, Doctoral Thesis, Computer Science Division, University of Calif., Berkeley, 1994.

[15] Emiris I. and Canny J., A Practical Method for the Sparse Resultant, *Proc. ACM ISSAC*, 183–192, Kiev, July 1993.

[16] Emiris I. and Canny J., Efficient Incremental Algorithms for the Sparse Resultant and the Mixed Volume, Technical Report, *Univ. of California, Berkeley*, 1994.

[17] Faugère J.C., Gb Reference Manual, PoSSo project, *Institut Blaise Pascal*, France.

[18] Faugère J.C., Gianni P., Lazard D., and Mora T., Efficient Computation of Zero-Dimensional Gröbner Bases by Change of Ordering, *Journal of Symbolic Computation*, 1992.

[19] Gantmatcher F.R., *Matrix Theory*, vol. 1. Chelsea Publishing, 1959, Chap. 2.

[20] Gao, X.-S., Wang, D.-K., On the Automatic Derivation of a Set of Geometric Formulae. *Mathematics Mechanization Research Preprints* No. 8, 1992, 26-37, Academia Sinica, China.

[21] Gerber, L., The Orthocentric Simplex as an Extreme Simplex. *Pacific Journal of Mathematics* 56, 1975, 97-111.

[22] Grigoryev D.Yu. and Chistov A.L., *Sub-exponential Time Solving of Systems of Algebraic Equations.* LOMI Preprints E-9-83 and E-10-83, Leningrad, 1983.

[23] Gelfand I.M., Kapranov M.M. and Zelevinsky A.V., *Discriminants, Resultants and Multidimensional Determinants*, Birkhäuser, Boston, 1994.

[24] Herstein I.N, *Topics in Algebra*, Blaisdell, Waltham MA, 1964.

[25] Heymann, W. Problem der Winkehalbierenden *Ztschr. f. Math. and Phys.* No. 35, 1890.

[26] Hoffman, C.M., Algebraic and Numeric Techniques for Offsets and Blends, *Computations of Curves and Surfaces*, W. Dahmen, M. Gasca and C. Micchelli, eds., Kluwer Academic Publishers, 1990.

[27] Kapur D. and Lakshman Y.N., Elimination Methods: an Introduction. *Symbolic and Numerical Computation for Artificial Intelligence* B. Donald et. al. (eds.), Academic Press, 1992.

[28] Kapur D., Lakshman Y.N. and Saxena T., Computing Invariants using Elimination Methods, *IEEE International Symposium on Computer Vision*, Coral Gables, Florida, USA, November 1995.

[29] Kapur D., Saxena T. and Yang L., Algebraic and Geometric Reasoning using Dixon Resultants, *Proc. ACM International Symposium on Symbolic and Alge-*

braic Computation, Oxford, England, July 1994.

[30] Kapur D and Saxena T., Comparison of Various Multivariate Resultant Formulations, *Proc. ACM International Symposium on Symbolic and Algebraic Computation*, Montreal, July 1995.

[31] Kapur D and Saxena T., Sparsity Considerations in the Dixon Resultant Formulation, *Proc. ACM Symposium on Theory of Computing*, Philadelphia, May 1996.

[32] Macaulay F.S., *The Algebraic Theory of Modular Systems*, Cambridge Tracts in Math. and Math. Phys., 19, 1916.

[33] Manocha D., *Algebraic and Numeric Techniques for Modeling and Robotics*, Doctoral Thesis, Computer Science Division, University of Calif., Berkeley, 1992.

[34] Michel J.D., Nandhakumar N., Saxena T. and Kapur D., Using Elimination Methods to Compute Thermophysical Algebraic Invariants from Infrared Imagery, *13th National Conference on Artificial Intelligence* (AAAI 96), Portland, Oregon, USA, August 1996.

[35] Morgan A.P., *Solving Polynomial Systems Using Continuation for Scientific and Engineering Problems*, Prentice-Hall, Englewood Cliffs NJ, 1987.

[36] Morgan A.P. and Sommese A.J., A Homotopy for Solving General Polynomial Systems that Respect m-Homogeneous Structures, *Appl. Math. Comput.*, 24:101-113, 1987.

[37] Ritt J.F., *Differential Algebra*, American Mathematical Society, New York, 1950.

[38] Ronga F. and Vust T., Stewart Platforms without Computer, *Manuscript*, 1992.

[39] Sederberg T. and Goldman R., Algebraic Geometry for Computer-Aided Design, *IEEE Computer Graphics and Applications*, Vol. 6, No. 6, pp. 52-59, 1986.

[40] Sturmfels B., Sparse Elimination Theory, *Proc. Computat. Algebraic Geom. and Commut. Algebra*, D. Eisenbud and L. Robbiano, eds., Cortona, Italy, June 1991.

[41] Sturmfels B. and Zelevinsky A., Multigraded resultants of the Sylvester type, *Journal of Algebra*, 1992.

[42] Verschelde J., Verlinden P., and Cools R., Homotopies Exploiting Newton Polytopes for Solving Sparse Polynomial Systems, *SIAM Journal of Numerical Analysis*, 31(3):915-930, 1994.

[43] Weyman J. and Zelevinsky A., Determinantal Formulas for Multigraded Resultants, *Journal of Algebraic Geometry*, pp 569-597, 1994.

[44] Wu W.T., Basic Principles of Mechanical Theorem Proving in Elementary Geometry, *Journal of Automated Reasoning*, 2, 219-252, 1986.

[45] Zippel R. *Effective Polynomial Computation*, Kluwer Academic Publishers, Boston, 1993.

Appendix A

Benchmark Polynomial Systems

Following are all the 14 polynomial systems which were used as benchmarks to compare the various resultant formulations in chapter 6. Each example gives a brief description of the origin of the polynomial system, the variables to be eliminated, and the polynomials themselves.

1. Side-Bisector Relation for a triangle. Described in section 7.2.1.

2. Maximum Volume of a Tetrahedron. Described in section 7.2.1.

3. Distance of intersection of two conics from the origin. Eliminate x, y from:

$$
\begin{aligned}
a_1x^2 + b_1xy + c_1y^2 + d_1x + e_1y + f_1 &= 0 \\
a_2x^2 + b_2xy + c_2y^2 + d_2x + e_2y + f_2 &= 0 \\
x^2 + y^2 - T &= 0
\end{aligned}
$$

4. Conditions for perpendicular intersection of a conic and a circle. Eliminate x, y from:

$$\begin{aligned}
ax^2 + bxy + cy^2 + dx + ey + f &= 0\\
x^2 + y^2 + d_1x + e_1y + f_1 &= 0\\
\frac{dq_1}{dx}\frac{dq_2}{dx} + \frac{dq_1}{dy}\frac{dq_2}{dy} &= 0
\end{aligned}$$

5. Four random polynomials (looks very similar to Aronberg's system). Eliminate x, y, z from:

$$\begin{aligned}
a_1x + a_2y + a_3z + a_4 &= 0\\
b_1xy + b_2yz + b_3zx + b_4 &= 0\\
c_1xyz + c_2 &= 0\\
d_1xyz + d_2x + d_3y + d_4z &= 0
\end{aligned}$$

6. Implicitization of a sphere. Eliminate r, s from:

$$\begin{aligned}
x(r^2 + 1) - (r^2 - 1) &= 0\\
y(r^2 + s^2)(r^2 + 1) - 4r^2s &= 0\\
z(r^2 + s^2)(r^2 + 1) - 2r(r^2 - s^2) &= 0
\end{aligned}$$

7. Implicitization of a bicubic surface. Described in section 7.2.2.

8. Implicitization of a dense bicubic surface parameterization. Eliminate s, t from:

$$\begin{aligned}
& t^3(-85s^3 - 55s^2 - 37s - 35) + t^2(97s^3 + 50s^2 + 79s + 56) \\
& \quad + t(49s^3 + 63s^2 + 57s - 59) + 45s^3 - 8s^2 - 93s + 92 \\
& - x(t^3(-53s^3 + 85s^2 + 49s + 78) + t^2(17s^3 + 72s^2 - 99s - 85) \\
& \quad + t(-86s^3 + 30s^2 + 80s + 72) + 66s^3 - 29s^2 - 91s - 53) = 0
\end{aligned}$$

$$\begin{aligned}
& t^3(43s^3 - 62s^2 + 77s + 66) + t^2(54s^3 - 5s^2 + 99s - 61) \\
& \quad + t(-50s^3 - 12s^2 - 18s + 31) - 26s^3 - 62s^2 + s - 47 \\
& - y(t^3(-53s^3 + 85s^2 + 49s + 78) + t^2(17s^3 + 72s^2 - 99s - 85) \\
& \quad + t(-86s^3 + 30s^2 + 80s + 72) + 66s^3 - 29s^2 - 91s - 53) = 0
\end{aligned}$$

$$\begin{aligned}
& t^3(-91s^3 - 47s^2 - 61s + 41) + t^2(-58s^3 - 90s^2 + 53s - 1) \\
& \quad + t(94s^3 + 83s^2 - 86s + 23) - 84s^3 + 19s^2 - 50s + 88 \\
& - z(t^3(-53s^3 + 85s^2 + 49s + 78) + t^2(17s^3 + 72s^2 - 99s - 85) \\
& \quad + t(-86s^3 + 30s^2 + 80s + 72) + 66s^3 - 29s^2 - 91s - 53) = 0
\end{aligned}$$

9. Four random second degree polynomials. Eliminate r, s, t from:

$$-2rs - 3ry + rz + sz - 3t^2 - 3tx - 2w^2 + 3tz - wy = 0$$

$$
\begin{aligned}
1 + ry - 2tx + 3w^2 - tz - r^2 - sx - y^2 &= 0 \\
-r - rs + 2ry + 2sx + y^2 + ty - 3wx - 3xy &= 0 \\
3s + tz + 2ty - 3rt - rw + 3sw + wz + 2xz &= 0
\end{aligned}
$$

10. Five Random third degree polynomials. Eliminate q, r, s, t from:

$$
\begin{aligned}
3rwy + q^3 + 3rx + 8stw &= 0 \\
-4s^2w + 4q^2t - 8qrw + 8qst &= 0 \\
7 - 8rsx + 9qrt &= 0 \\
r^2 - 8qsy + 7qrs + 4twy &= 0 \\
9q^2t - qrx + 9qsx &= 0
\end{aligned}
$$

11. Cubic Surface Implicitization. Eliminate s, t from:

$$
\begin{aligned}
t^3 - 3st - s^3 - s + x &= 0 \\
ts^2 - 3t + 1 - y &= 0 \\
2t^3 - 5st + t - s^3 - z &= 0
\end{aligned}
$$

12. Equilibrium of a Lorentz system. Eliminate x, y, z from:

$$
\begin{aligned}
y(z-t)-x-a &= 0\\
z(t-x)-y+a &= 0\\
t(x-y)-z+a &= 0\\
x(y-z)-t+a &= 0
\end{aligned}
$$

13. Camera Motion from Point Matches. Described in section 7.1.2.

14. Conformal Analysis of Cyclic Molecules. Eliminate t_1, t_2 from:

$$
\begin{aligned}
-13-t_2^2-t_3^2-t_2^2t_3^2+24t_2t_3 &= 0\\
-13-t_1^2-t_3^2-t_1^2t_3^2+24t_1t_3 &= 0\\
-13-t_2^2-t_1^2-t_2^2t_1^2+24t_2t_1 &= 0
\end{aligned}
$$